Representation of Knowledge for Chess Endgames Towards a Self-Improving System

Thesis

Representation of Knowledge for Chess Endgames: Towards a Self-improving System

A thesis submitted for the degree of

DOCTOR OF PHILOSOPHY

in the

FACULTY OF MATHEMATICS

of

THE OPEN UNIVERSITY

by

MAX ARTHUR BRAMER, B.Sc.

March 1977

Contents

Tables

Dr M.A. Bramer HDB 5001 PhD

Internal Memorandum

From M A Bramer, S.C.S. MB/PAB/649/-

To M Sullivan, Higher Degrees

Subject Ph.D Thesis Date 6 July 1977

I am writing to confirm that I am willing for the library's copy of my thesis to be made available both for reading and for photocopying at the discretion of the Librarian.

M. A. Bramer

M A Bramer

Abstract

This thesis describes an investigation of the problems involved in representing knowledge within the task area of elementary Chess endgames. Two major criteria are taken for the choice of a model of the chessplayer's knowledge : firstly, that algorithms constructed using the model should be natural from the viewpoint of a chessplayer and commensurate with his view of the complexity of the task, and secondly, that the algorithms should be capable of refinement in the light of experience in a manner which preserves the previous property.

Elementary chess endgames are studied as a field in which programs based on tree-searching and traditional evaluation functions have achieved poor results and where tree-searching seems to play little or no part for people. It is therefore possible to examine problems of knowledge representation and program refinement largely independently of the tree-searching paradigm.

A long term aim of the research is to develop a representation suitable as the basis for a fully automatic system of algorithm refinement, whilst maintaining the criteria given above.

A model is proposed and algorithms are given for two endgames, King and Rook against King (KRK) and King and Pawn against King (KPK) using this model. It is argued that both algorithms are reasonably natural and compact representations and experiments in refining these algorithms are described in detail. In both cases, the process of refinement is shown to be a reasonably straightforward one (for people) and one which maintains the properties of naturalness and compactness. The possibility of automating this process is considered.

Acknowledgements

The work described in this thesis was performed under the supervision of Mike Pengelly, Professor of Computer Science at the Open University, and Michael Clarke of Queen Mary College, London, to both of whom I am grateful for their advice and support at all stages of the project. My further thanks are due to the latter for the use of his King and Pawn against King database and to the former, in his role as Dean of the Faculty of Mathematics, for his continuing efforts to establish a satisfactory research environment at the Open University.

Considerable use has been made of the computing facilities of both Queen Mary College Computer Centre and the Student Computing Service of the Open University and I am grateful to both departments not only for the computer time but also for the support and forebearance of their staff.

The final stages of the work were partly supported by a grant from the Research Funding Sub-committee of the Open University, which made it possible for me to travel to conferences and meetings and purchase important reference books.

I should also like to thank Vivienne Yarwood and Jackie Miller for their painstaking and patient work in typing (and retyping) this thesis.

Finally, I am indebted to my wife Dawn for uncountably many discussions of both the content and the presentation of the material that follows. Her assistance took a particularly tangible form in the preparation of the many diagrams and in the methodical checking of the typescript, as well as editorial advice on its final form. More importantly, without her continual enthusiasm and encouragement it is unlikely that this work would ever have been completed.

1. Introduction

The research to be described in this thesis is a project in the general field of Artificial Intelligence, concerned with the representation of knowledge for endgames in the game of Chess, with particular reference to the possibility of eventually developing a self-improving system to play such endgames. It is appropriate, therefore, to begin by discussing the meaning and significance of some of these terms and the reason for the choice of project.

Artificial Intelligence

In a recent paper, Howe et al (1975) describe Artificial Intelligence as "the study of how to organize processes to exhibit intelligent behaviour" and give as a central theme the importance of the way knowledge is represented and used.

In the absence of a satisfactory definition of "intelligence", a reasonable operational definition of an intelligent program might be one which has a level of performance which if exhibited by a human would be generally considered to indicate the use of intelligent reasoning, as opposed to powers of calculation alone. Thus programs to invert matrices, calculate square roots, etc. are excluded, but programs to play Chess at master level, discover new Mathematical theorems, recognize handwriting etc. are included.

It is not required that the program use methods identical in every respect to those employed by its human counterparts but it is probable that in many cases a study of such human methods may prove to be the most fruitful line of investigation.

Chess-playing programs

Of the many tasks with which Artificial Intelligence has been concerned, constructing a chess-playing program to perform at the level of a human expert has been one of the most persistent and also one of the most elusive.

The idea of a "Chess-playing machine" seems to have a particular fascination, pre-dating both Artificial Intelligence and the electronic computer itself. Shannon's classic paper "Programming a computer for playing Chess" was published in 1950, but an electro-mechanical device for playing the endgame King and Rook against King was demonstrated at the Paris World Fair of 1900. Indeed, an automatic chessplayer - a lifesize figure dressed as a Turk and seated at a large box which served as a playing table - was demonstrated to Empress Maria Theresa of Austria approximately two centuries ago and later also to Empress Catherine II of Russia. "Von Kempelen's Turk" was, in fact, a hoax operated by a human chessplayer inside the box but the idea was of sufficient public interest to attract large audiences throughout both Europe and America for almost seventy years (Lasker, 1959). Newell, Shaw and Simon (1958) justify their interest in Chess from the Artificial Intelligence viewpoint as follows.

"Chess is the intellectual game par excellence. Without a chance device to obscure the contest, it pits two intellects against each other in a situation so complex that neither can hope to understand it completely, but sufficiently amenable to analysis that each can hope to outthink his opponent. The game is sufficiently deep and subtle in its implications to have supported the rise of professional players, and to have allowed a deepening analysis through 200 years of intensive study and play without becoming exhausted or barren. Such characteristics mark Chess

as a natural area for attempts at mechanization. If one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavor".

Compared with tasks such as proving theorems in predicate logic, Chess has the substantial advantage of an extensive culture, comprising thousands of published books and articles, centuries of theoretical development and possibly millions of active practitioners throughout the world.

Without this background culture it is likely that existing chess-playing programs would be considered as outstanding examples of the success of Artificial Intelligence techniques.

Instead, it is evident that a peak of performance is being approached, perhaps already reached, which is well short of expert by human terms. By including encyclopaedic knowledge of openings and making use of the computer's power of infallible calculation (without sudden lapses of memory or concentration) a level of performance can be achieved which at best is comparable to medium English club standard, with endgame play which would frequently be an embarrassment even to a beginner, where endgames are drawn or even lost despite a large material advantage.

The small improvement which has been achieved in the performance of the best programs over the last ten years suggests that valuable though the initial pioneering work has been, the tournament performance of existing programs (see, for example, Hayes and Levy (1976)), should be looked at primarily in terms of providing a "benchmark". It would seem that for significant progress to be made, new techniques are now required.

Virtually all programs to play the complete game of Chess make use of the "tree-searching and evaluation function" model proposed by Shannon.

This model is superficially attractive since chessplayers certainly do make use of analysis to a greater or lesser extent and it is easy to prove that given sufficient time (which Shannon calculates as at least 10^{90} years in the initial position) the best move in every position can be found by analysis to "terminal positions" alone.

For practical reasons, program improvements have generally been concerned with developing more powerful tree-searching techniques to allow an increasingly deep level of analysis to be performed.

The problem of representing the strong player's knowledge of the game has frequently been greatly underestimated. When recognised, it has invariably proved intractable.

Perhaps inevitably, then, chess-playing programs provide an example of a "solution" which has been developed to suit the overt capabilities of the computer, rather than the requirements of the problem itself.

To give an illustration of the essential weakness of this approach, an expert player may evaluate a position (either the "current" position or one which occurs at the end of an analysed variation) as favourable to himself because the opponent has a slightly vulnerable Queen side and a poorly placed dark-squared Bishop. Against this he may have a weak King pawn which he judges to be of lesser importance. The expert's judgement is based not on analysis, although it will often need to be supplemented by detailed analysis, but on a general understanding of the game derived from some combination of past experience and study. This

understanding is virtually unquantifiable by conventional means and could certainly not be replaced by an improved power of analysis.

Representation of Knowledge

To make a significant improvement to the standard of chess-playing programs it is probably unnecessary to develop existing tree-searching techniques any further (since they may already be better than is really needed). The problem which remains to be solved is how can the expert chessplayer's knowledge of the game be effectively represented in a computer program. Although it may be expected to be an extremely complex one, this problem of the representation of knowledge is central not only in the context of Chess, but (as Howe et al. point out) to Artificial Intelligence research as a whole.

For a given body of knowledge, there are, in principle, many different representations which can be chosen and embodied in computer programs. In one sense, all of these representations are equally valid. However, in practice, it is almost invariably the case for complex "intelligent" tasks such as playing Chess that the necessary knowledge required is not fully available to the programmer. There is then an important criterion for evaluating alternative representations: their capacity for modification.

Just as human knowledge is not static, but evolves in the light of experience, so should it be possible for the knowledge embodied in a computer program to be modified when deficiencies become apparent. All programs can, of course, be changed. What is important is the ease with which such changes can be made and the extent to which the program's performance indicates the nature of the change required.

The more incomplete or incorrect the specific knowledge embodied in the program, the more crucial is the choice of representation. With a poor choice, it is likely that an improvement to one part of a program will result in errors arising elsewhere, whereas human learning is, in general, a process of continual improvement with occasional lapses due to over-generalization, etc.

A second criterion for evaluating alternative representations is that the size and complexity of the corresponding algorithms should be commensurate with the problem, as judged in human terms. Naturally, such judgements can only be made within broad limits. In order to satisfy this criterion for "intelligent" tasks, it would seem to be necessary to concentrate on general rules rather than specific instances. Thus, in the case of Chess, it would not be satisfactory to treat each position in which the program performed badly as a "special case", since specific positions rarely recur in Chess games. It would seem to be necessary to treat each such position as a representative of some wider class of positions, sharing some common feature.

Self-improving programs

In the previous discussion, it has been assumed that program improvements will be performed by the programmer in the usual way.

In fact, however, it is clearly advantageous for a program to be able to improve its own performance on the basis of its experience. Thus the idea of an "intelligent" program could be extended to combine the performance of an intelligent task with the capacity for self-improvement.

Learning programs have been written for the solution of a number of problems in Artificial Intelligence, the most successful of these

probably being Samuel's programs for playing Checkers (Samuel (1959) and (1967)). Learning programs for several other games have also been written, including partnership dominoes (Smith (1973)) and Go-Moku (Murray and Elcock (1968)). The methods employed are in most cases specific to the particular problem to be solved and none of the techniques used would appear to be directly transferrable to the game of chess, except in very restricted situations, because of the much greater complexity of the game.

Here again, the choice of a representation is likely to be critically important.

For self-improvement to be a practical possibility, it must be possible for the elements comprising the algorithm to be manipulable by a higher-level program.

Endgames

The research described in this thesis is concerned with representing the strong player's knowledge not of the middlegame but of the endgame, with particular examples taken from "elementary" endgames. There are a number of reasons why an investigation of endgames has been chosen in preference to middlegames.

Firstly, it is in the endgame that the inadequacy of the "tree-searching and evaluation function" model becomes most apparent.

As pointed out previously, programs written to play the full game of chess almost invariably make use of this model and play extremely poorly in the endgame. This is itself a reflection of the greatly reduced part which analysis plays in the endgame compared with an awareness of the

significance of certain key patterns of pieces. For elementary endgames, detailed analysis seems to play little or no part at all.

For more complicated endgames there is an inevitable trade-off between the number of patterns which are recognized and the amount of detailed analysis which needs to be performed, depending on the skill of the player, but the balance of importance is strongly in favour of the former and it is safe to say that the grandmaster's superiority in endgame play over players of lesser ability consists far more of a more extensive knowledge of significant patterns than of greater analytical powers.

It was perhaps considerations of this sort which prompted Shannon to point out that a different model would be necessary for the endgame, than for the middlegame, although this advice has almost invariably been ignored in programs to play the full game of chess.

A second advantage of investigating endgames rather than middlegames is that in many cases substantial theoretical knowledge is readily available in textbooks. Some endgames are even thought to be completely understood theoretically, for example King and Pawn against King, whereas middlegame theory is only partially understood and subject to change. Moreover it is possible to consider each endgame in isolation, whereas the different strategic features of a middlegame position will inevitably interact, with a corresponding increase in complexity. Thus an expert knowledge of the strengths and weaknesses associated with an isolated Queen Pawn in the middlegame will prove worthless unless combined with other knowledge.

The existence of a substantial body of endgame knowledge not only provides a starting point for investigation but also provides a

background against which the performance of a given program can be evaluated.

In the middlegame poor play will often escape attention and may still lead to a win. In the endgame poor play will usually be easily seen as such and will often prove disastrous.

The net effect of the above considerations is that a study of the endgame offers the opportunity to investigate the problems involved in representing the expert's chess knowledge in a relatively simple context where they can be examined independently, or almost independently, of the problems associated with efficient tree-searching.

Although the particular representations developed for the endgame may not be directly applicable to the middlegame, it is to be hoped that the general principles on which they are based may be transferrable to the middlegame and to other problems in the broader area of Artificial Intelligence.

Michie (1976a) describes Artificial Intelligence as seeking "general methods of bringing domain-specific aids to the assistance of general algorithms". From the viewpoint of generalizability, the domain-specific knowledge associated with a particular endgame is of little significance. What is important is the overall framework into which this knowledge is fitted.

Endgame - playing programs: the state of the art

Compared with the large number of computer programs which have been written to play a complete game of chess, relatively little attention has so far been devoted to investigating the problems of programming the endgame.

It is some measure of the difficulty involved in programming the endgame that virtually all "purpose-built" programs have been written only for endgames which are elementary by human standards (in the sense that they are discussed only in textbooks intended for beginners or inexperienced players) and that a variety of different representations have been tried. Programs specifically for endgames have been written by Huberman (1968), Tan (1972, 1974) and Zuidema (1974). In addition, a major project in this area is currently being pursued under the directorship of Donald Michie of Edinburgh University. These approaches can be broadly classified as either "structural" or "procedural", depending on whether the domain - specific knowledge is held in the form of a database, independent of the general solution algorithm, or embedded in the procedures of the algorithm itself. The distinction between the two categories is not always clear cut and a combination of approaches is also possible; nevertheless the distinction is helpful. In general, a procedural algorithm is easier to construct and is computationally more efficient, but is considerably more difficult to modify and to relate to the subject expert's own understanding of the task. Constructing a self-modifying system based on a procedural representation would seem to be impossible. Since this research is concerned with representing knowledge of the endgame in a form which is meaningful to the chessplayer himself and which permits the underlying algorithm to be modified in a simple and natural way - ultimately by the computer system itself - a structural approach has been adopted.

As will be seen from the discussion in Section 3, the problem which has so far proved insurmountable in practice (using either approach) is to produce an algorithm for a particular endgame, which is remotely commensurate with the complexity of that endgame, as judged by the amount of explanation devoted to it in a typical chess textbook. This would seem to be an important criterion for evaluating any proposed representation of the chessplayer's knowledge.

In most cases, the methods used do not appear to be capable of extension to other endgames and almost invariably the component elements of the algorithms themselves bear little overt resemblance to the features of the endgames which a chessplayer might consider significant, with the result that it would not be possible for a chessplayer to determine the algorithms' moves or to judge the likely quality of their performance, except by examining the play produced over a series of games.

These comments are, of course, a reflection of the difficulty associated with the problem. As is generally the case for game-playing programs, there is only a small amount of information available about the testing which has been applied to endgame-playing programs and an absence of quantifiable results concerning their performance.

The aims of this research

The preceding discussion provides the background to the research to be described in this thesis.

The aims of this research can be summarized as follows.

(1) To devise a structural model for representing endgame knowledge which satisfies as well as possible the criteria developed above, that is, it should be applicable to a reasonably wide range of different endgames and should enable algorithms to be written which are commensurate in terms of size and complexity with the complexity of the corresponding endgames, as judged by the explanation devoted to them in standard chess textbooks.

The algorithms should incorporate elements which can be closely related to the significant features of particular endgames, as they may be expected to be seen by chessplayers, thus enabling them to be examined and evaluated (and ultimately, perhaps, even written) by chessplayers themselves. The algorithms should be capable of relatively straightforward modification (by the programmer) in the

light of their own performance and the changes required should be indicated by the deficiencies observed in the performance. Program modification should generally involve a continual improvement to the overall performance without the necessity to introduce a large number of "special cases".

The model should be such as to allow the possibility of the eventual development of a self-improving or learning system. Thus the elements of the algorithms produced should be capable of manipulation by a "higher-level" program.

(2) To demonstrate the applicability and use of the model by constructing algorithms for specific endgames.

Since one aim is to investigate the problems of representing knowledge independently of the problems of efficient tree-searching, two "elementary" endgames are chosen as examples, for both of which it turns out that tree-searching seems to be unnecessary altogether. An example of the kind of algorithm which would be required for part of a more complex endgame is given as an appendix.

(For this purpose it is irrelevant whether or not the algorithms produced are perfect or optimal in any sense, the correspondence between the algorithms and the textbook descriptions from which they are derived is of principal importance.)

(3) To demonstrate the means by which the algorithms can be modified by the programmer in response to deficiencies in their play revealed by games against human opponents and the degree of difficulty involved in this process. As mentioned in (1) above, the ease and "naturalness" of program modification are major criteria when evaluating the overall model.

(4) To investigate the problems involved in constructing a self-improving system.

For this purpose, the task was chosen of improving (manually) a program for one specific endgame to play perfectly in every one of a subset of the possible positions in that endgame, with the intention of synthesizing some of the features which would be required in a self-improving system. Such a system was not itself constructed.

Although this aim is of importance in itself, it also bears strongly on the question of evaluating the choice of model proposed.

(5) As a subsidary aim, to produce quantifiable results wherever possible. Here again, concentrating on elementary endgames is likely to be helpful.

Outline of this Thesis

The model itself is described in detail in Section 2, and those used in previous work in the area of chess endgames are discussed in Section 3. In Sections 4-7 an algorithm for the endgame King and Rook against King is developed and used as the basis for further investigation. An initial algorithm is given in Section 4, based on a detailed analysis of the winning strategy for the stronger side. Some of the practical aspects of using the model are also considered at this stage, in particular the problems which arise from symmetry.

Some of the conceptual problems associated with testing an algorithm of this type are discussed in Section 5. To enable the algorithm to be

carefully evaluated in practice, a program was set up to play the endgame competitively against human opponents and a substantial number of games were played. This empirical testing is described in Section 6 with a discussion of the changes which were judged to be necessary to the original algorithm, and their rationale. The revised algorithm is given in tabular form followed by a discussion of its basic components (primitives).

The overall significance of the results of this testing and the changes found to be necessary to the original algorithm are considered in Section 7, together with an evaluation of the performance of the revised program and some illustrative games played against human opponents.

The second part of the thesis concentrates on an algorithm for a second endgame, King and Pawn against King, which is considered principally from a different viewpoint: what problems and techniques are involved in modifying an algorithm (based on the model described in Section 2) to play perfectly and what elements would be required in a self-improving program to perform such a process automatically. An initial form of the algorithm is given in Section 8, followed in Section 9 by a general description of the background to and the significance of the experimentation which is discussed in detail in Section 10. Section 11 summarizes some of the requirements for a self-improving program.

Overall conclusions and possibilities for further research are presented in Section 12.

A "non-technical" description of the first version of the King and Rook against King algorithm (as described in Section 4) is given in

Appendix 1, in a form designed to demonstrate the distinction between the overall model and the endgame - specific additional information which needs to be supplied.

Some steps towards the development of an algorithm for a more complex ending, King, Rook and Pawn against King and Rook are described in Appendix 2. The final appendix contains a summary of the notation used in the text And is followed by the references.

Lastly, reprints are included of two technical reports on work tangential to that described in this thesis, which were issued by the Mathematics Faculty of the Open University.

The first, "King and Pawn against King : Some quantitative data", gives a variety of technical data about that endgame. The second, "King and Pawn against King : using effective distance", contains an implementation of the predicate function "Pawn can run" (advance to the queening square without being captured) based on the idea of the "effective distance" (length of the shortest permissible route) between two squares.

2. The Model

In this section a model for representing chess endgame knowledge, which has been designed to satisfy as closely as possible the criteria developed in Section 1, is described and discussed.

This model is used as the basis for all the work subsequently described in this thesis.

Since a major objective is to devise a representation of the chessplayer's endgame knowledge which is meaningful to the chess player himself, it is appropriate to begin by considering the information presented to the reader of a typical chess textbook. Taking this information as a starting point is also likely to help ensure that any eventual computer program corresponds reasonably closely to the strategy chosen by the human chessplayer and is thus of commensurate complexity with the problem as judged in human terms.

A textbook discussion of an "elementary" endgame generally occupies no more than two or three pages and comprises a few general "rules of play" accompanied by some example variations from diagrammed positions. The rules are normally only imprecisely worded and omit a number of important details which have to be inferred by the reader from the variations given. Nevertheless, this information is considered to be a complete and sufficient explanation for the reader and it is intended to have essentially the same meaning for every reader of the text. Clearly the interpretation put on the information given in the text by the reader serves to fill in the gaps "between the lines" of what is written. It would appear that the reader has some overall "conceptual framework" in terms of which he interprets the material that he reads. An important element in this framework is the ability to generalise from specific examples to general principles in accordance with some

unspecified series of rules.

The basic unit of information given to the reader of the endgame text is the diagrammed position and associated "best move". The reader is expected to generalize from each particular position given to a broader set of positions which are equivalent in some way to the original. Equivalence may be defined in a number of ways. For instance, it may involve the movement of the total configuration of pieces, say, one or two ranks up or down the board. The reader not only has an awareness of what constitutes an equivalent position but also a corresponding awareness of what does not, that is, he is able to recognise exceptions to his generalizations. Thus, for example, a position which would otherwise be equivalent, except for an "accidental" stalemate, is recognised as a totally different situation. Implicit in this is the idea of an "ordering" of objectives. In cases where an exception is difficult to recognise (e.g. the effect of a Rook Pawn in a Pawn endgame) it will generally be given as a separate example in its own right. When choosing a move in a given "initial" position, the primary consideration is what can best be achieved from that position rather than the specific configuration of the pieces themselves. Thus when choosing a move for White, it would seem to be necessary to make comparisons between the possible "successor positions" with Black to move and <u>vice versa</u>.

The model to be described makes use of the idea of classes of equivalent positions with a particular side, say Black, to move which are employed when finding moves for the <u>other</u> player (White).

Following the preceding discussion it may be said that the model corresponds to the underlying conceptual framework in terms of which the reader of a textbook comprehends and evaluates the material presented.

Looking at the situation in another way, the computer and the model together comprise an "endgame playing machine" which can be "programmed" for a particular ending by providing a relatively small amount of specific information.

2.1 Defining the problem

As a basis for the discussion which follows, it will be assumed that the problem is to construct an algorithm to find the best move in any legal position (for a particular endgame) with one chosen side to move.

For convenience this side will be referred to throughout as White. (Note that to find moves for both sides would require two separate algorithms, one for finding White moves and one for Black.) Although the discussion which follows is completely general, examples from the King and Rook against King endgame are used for illustration where necessary. In this case White is always taken to be the side with the Rook since this side has the more interesting and complex strategy.

It should be noted in passing that no attempt has been made to cater explicitly for either the "three-fold repetition of position" drawing rule or the fifty-move rule. If either of these conditions were to arise in an endgame (such as King and Rook against King) where a draw would be unacceptable to White, the move-finding algorithm should simply be considered to have failed.

The following notation will be used throughout the remainder of this thesis:

W - the set of all legal positions with White to move for a given endgame.

B - the set of all legal positions with Black to move for a given endgame.

p - an 'initial' position, with White to move, in which the best move is to be found ($p \in W$).

Q(p) - the set of all <u>immediate successors</u> of p, that is the positions (with Black to move) which arise after a legal White move in p ($Q(p) \subset B$).

Q(p) is to be considered as an <u>ordered</u> set, although the particular ordering rule chosen is unimportant.

A suitable rule is to take the order in which the positions are generated for some particular implementation.

$q_1(p)$, $q_2(p)$, ... - the successors of p themselves ($q_1(p) \in B$ etc.),

where $Q(p) = \{q_1(p), q_2(p), \ldots\}$

This notation will be simplified when no confusion can arise, thus Q(p) will often be abbreviated to Q and $q_1(p)$ to q_1 or q(without a subscript).

Further notation will be introduced as and when necessary. A summary of all notation used will be given in Appendix 3.

2.2 <u>An overview of the model - equivalence classes of positions and class value</u>.

The fundamental principle underlying White's play can be summarised as "in any position p, choose the move which gives the most favourable successor position".

This is to be interpreted as meaning that the successor positions are to be generated and compared with each other and the most favourable one (for White) chosen.

A general means of comparing positions with Black to move must therefore be available and since this is independent of the choice of any specific initial position p, it follows that the successor positions must be compared solely on their own merits, without any reference to p. This point will be discussed further in Section 2.9.

In the basic form of the model, which is described below, no tree-searching is performed below the level of the immediate succesors of a given initial position. For the two elementary endgames discussed in this thesis, this form of the model seems to be sufficient, which accords with the observation that the chessplayer appears to make little or no use of analysis when playing elementary endgames, relying instead on his knowledge of significant patterns.

Although the basic form of the model is of general applicability, a practical implementation of an algorithm for a complex endgame may require a more or less substantial amount of tree-searching to supplement the pattern matching of the basic model. In Section 2.8, an extended form of the model is described which enables tree-searching to be incorporated in a readily controllable way.

However, for the two basic algorithms discussed in this thesis, only the basic form of the model appears to be necessary, and this form is described in the remainder of this and the following five sections.

Assuming that a complete ordering, or ranking, can be placed on the set B (where the highest ranked position is the one most favourable to White), the move-finding algorithm for a given initial position p is as follows:

(a) generate the set Q(p) of successor positions;

(b) find the highest ranked member of Q(p), say $q_i(p)$;

(c) play the move for White corresponding to $q_i(p)$.

For any endgame it will be meaningful to speak of a ranking of the members of B, provided that some suitable (arbitrary) criteria are adopted for resolving ties between equally favourable positions (e.g. between two positions where Black is checkmated).

As will be seen, the method used for ranking positions is essentially a two stage one. The basis of the method is a division of the set B of legal positions with Black to move into a number of subsets. All the positions in any subset are equivalent in the sense that they all satisfy the same membership criterion (a logical combination of predicate functions). The subsets are chosen to be disjoint and exhaustive, i.e. so that every position belongs to exactly one subset, and therefore constitute a <u>partition</u> of the set B into <u>equivalence classes</u>.

In what follows, the equivalence classes will be denoted by Class 1, Class 2, ... or $C_1, C_2, \ldots$, as convenient. If there are N classes, then

$$C_i \subset B \quad (i = 1,2,\ldots,N)$$

$$C_i \cap C_j = \emptyset \quad (i,j = 1,2,\ldots,N \; ; \; i \neq j)$$

$$C_1 \cup C_2 \cup \ldots \cup C_N = B.$$

The class of which a given position b with Black to move ($b \in B$) is a member will be represented by C(b).

There are, of course, many different ways in which a partition of B into equivalence classes can be made for any endgame. The essential point is that the classes must be specified in a way which enables a unique ranking to be made between them. Thus, given two classes, i and j, it must be possible to say either that every member of class i is better than every member of class j, or vice versa. In general this will only be possible when all the positions in a class share some common feature related to the endgame under consideration, such as the relative positions of two or more pieces (with suitable allowances for symmetry, board edge effects, etc., if necessary). The specifications of two typical equivalence classes might be "all positions where Black is stalemated" and "all positions where the Kings are in opposition and Black is not in check".

It should be noted, in passing, that the conditions for a position with Black to move to be a member of either of these classes can be defined without any forward analysis ("lookahead"), i.e. solely in terms of its static features.

Assuming that N equivalence classes have been defined, the integers 1,2,...,N (or any N distinct integers) can be used to denote the ranking between them. The integer corresponding to the ranking of a particular class is called its class value; the class most favourable from White's viewpoint has a class value of N, the least favourable has a class value of 1.

The notation V_i and V(b) will be used for the value of Class i and the value of C(b), b ϵ B, respectively. In practice, V_i (i = 1,2,...,N) is held in the i^{th} row of the first column of a two-dimensional matrix known as a value table. The use of the remaining columns of this table will be described in Section 2.5.

The three step move-finding algorithm given previously can now tentatively be expanded into the following:

(a1) generate the set Q(p) of successor positions, where Q(p) = $\{q_1(p), q_2(p), \ldots\}$;

(b1) determine the class to which each member of Q(p) belongs, i.e. $C(q_1(p))$, $C(q_2(p))$, etc;

(c1) find the corresponding class values $V(q_1(p))$, $V(q_2(p))$ etc.

(d1) play the move for White which leads to the successor position with the highest class value.

The algorithm in this form takes no account, however, of the possibility of a tie at step (d1). This is an important point which will be taken up in Section 2.4.

2.3 Specifying the equivalence classes

Each equivalence class is specified by means of a suitable predicate function. In the simplest form, if the function is true for a particular position q then q is a member of the corresponding class. There is, however, an important consideration which makes it desirable to modify this simple scheme.

It is essential to ensure that no position belongs to more than one class, and satisfying this condition often means that the definitions of the predicate functions become extremely complicated. As an example, consider the following series of predicate functions defining six equivalence classes for the King and Rook against King endgame. It will be assumed, for purposes of illustration only, that these classes are all that are required for that endgame. The predicate functions will be referred to as properties of a given position q.

Table 1 Equivalence classes : an example

Class	Property of position q (Black to move)	Class Value
1	Black is checkmated	6
2	Black is stalemated	2
3	Black can immediately capture White's Rook	1
4	The Kings are in vertical opposition, with Black in check along the rank	5
5	Black is not in check	4
6	(To be specified)	3

Class 1 is the most favourable to White (class value 6); Classes 2 and 3 are the least favourable (values 2 and 1, respectively). The final class should be ignored at present. As it stands, the specification of class 4 is incomplete. The property is intended to refer to positions where Black is checked by the Rook and thus forced to retreat a rank, a commonly occuring situation in this ending. As the property is specified, however, it is possible for q to belong not only to Class 4 but also to Class 1 (Checkmate positions) or Class 3 (positions where Black can capture the Rook), thus violating the principle that no position may belong to more than one equivalence class. To avoid this possibility, it would appear that the specification of class 4 must be extended to ensure that Black is not checkmated and cannot immediately capture the Rook. For similar reasons, the specification of Class 5 would have to be extended to exclude stalemate positions (Class 2) and positions where the Rook can be captured (Class 3).

In general, the specification of each class would need not only to describe some positive feature, such as those given in the table above, but actively to exclude the members of each of the other classes. For an endgame with any reasonably large number of classes this would inevitably lead to a complex set of specifications in which the basic underlying concepts (the 'positive' features of the positions) would be submerged. Fortunately, this problem can be overcome by adopting the following convention.

The classes are specified by means of their 'positive' features, as in the table above. The property corresponding to Class n is called <u>rule n</u>. The rules are arranged in the order in which they are to be evaluated and a given position q is said to belong to class n

if and only if rule n is satisfied by q and none of the preceding rules are satisfied. This convention automatically ensures that no position can belong to more than one class and enables the specification of each rule to be kept as simple as possible. It does, however, require the rules to be specified in the order in which they are to be tested. (In practice, there will generally be a certain amount of freedom; thus rules 1 and 2 could be transposed in Table 1. There may then be some marginal advantage in, for example, testing first the fule which is more frequently satisfied.)

The basic procedure for finding the class of which a given position q is a member is then to evaluate each rule in turn (working downwards from rule 1) until the first true rule is found. This rule then gives the class corresponding to position q. No further evaluation of rules is necessary. From now on, whenever the definition of a specific rule is discussed it will be assumed, unless otherwise stated, that the rule will only be applied to positions in which all the preceding rules are unsatisfied, i.e. false. This will, in turn, enable the detailed definitions of the rules themselves to be simplified in many cases. Thus, if classes 1 and 3 were transposed in Table 1, so that "Black can capture the Rook" were tested before "Black is checkmated" then in the definition of checkmate positions it would not be necessary to exclude the possibility of the Rook giving check from a square adjacent to Black's King. This could never occur since "Black can capture the Rook" must have been false.

The rules themselves are all predicate functions, the domain of each function being the set of all legal positions with Black to move in which all the preceding rules are false. "Rule" has been preferred to "predicate function" and other terms, such as

"pattern matching function" to avoid confusion with the associated functions described in Section 2.4.

The final rule in Table 1 has not so far been specified. Since it is essential for every position q to belong to some equivalence class, it will often be desirable to define the final rule in such a way that it must inevitably be satisfied by any position for which all the previous rules are false. This can be achieved by defining the rule to be always true, regardless of the position. Although the final class contains "residual" positions it will not necessarily be the lowest ranked class. Taking the example of Table 1, it is certainly preferable to stalemate positions (Class 2) and positions where the Rook can be taken (Class 3).

For some endgames it may even be that the final class will contain the large majority of positions, that is that the other classes will contain only a small number of relatively exceptional positions with the choice of the most favourable successor position being made between members of the final class.

2.4 Choosing between positions in the same class

As pointed out in Section 2.2, the provisional algorithm given there ignores the possibility of a tie at step (d1) "play the move which leads to the position with the highest class value".

In fact, however, if the number of classes is reasonably small, such ties will frequently occur. The method which is used to resolve them is important and plays a significant part in giving the algorithm "a sense of direction", as will be seen later in this section.

The class values specify a fixed ranking between equivalence classes; combining this with a procedure for ranking positions within the same class produces a complete ranking of all the members of B in order of favourability (from White's viewpoint). Thus it is possible not only to find the best move in position p, but to place all White's possible moves in order of preference. The procedure for ranking positions which are members of the same equivalence class (Class i) requires a number of associated functions or measures to be defined on the members of the class.

Whereas the rules defining class membership are predicate functions i.e. functions which take only the logical values true and false, the associated functions take numerical values, related to the positioning of the pieces on the board, such as the distance between the Kings or the number of the rank on which the King stands (the distance from the bottom edge of the board). The functions will, in general, vary from class to class. The number of associated functions for a particular class may be zero (since, for example, there is no non-arbitrary way of choosing between members of the class of positions "Black is checkmated"), but will generally be at least one.

If the functions corresponding to class i are f_{1i}, f_{2i}, f_{3i} etc., then positions in class i are ranked in the following manner. First, the value of function f_{1i} is evaluated for each position. This value is then used to place the positions in order of merit, that is, the positions are ranked in the order given by the values of f_{1i} (the highest ranked position has the largest corresponding value of f_{1i}, and so on). If there are any ties then the value of f_{2i} is evaluated for the tied positions, the position with the largest value of f_{2i} being ranked highest, followed by the position with the next largest value

etc., as before. Any remaining ties are resolved using function f_{3i} and so on until no ties remain or all the functions are exhausted. In the latter case ties are finally resolved arbitrarily.

A suitable means of making such a selection for members of Q(p) is to choose the <u>first</u> of the tied positions. It was for this reason that Q(p) was defined to be an <u>ordered</u> set.

Such "residual" ties may arise either when two positions are "identical" due to symmetry or, more often, when there is no advantage in forcing the algorithm to choose one position rather than the other, for example when both are positions in which Black is stalemated.

Given a complete ranking of all possible positions with Black to move and therefore of all the successors of any position p, it is a simple matter to find White's best move - the move which leads to the highest ranked successor position. Note that in practice it is only necessary to rank the successor positions in the highest occurring class, and to continue with the ranking procedure until the best position in that class is found.

The first associated function for class i (f_{1i}) is called the <u>primary function</u>. Subsequent functions (f_{2i},f_{3i}, ...) are called the <u>secondary function</u>, <u>tertiary function</u>, etc.

As previously stated, the use of associated functions tends to give the move-finding algorithm a "sense of direction". A situation which frequently occurs is that White has to make a choice between four or five moves each producing a position in the same equivalence class i (all other moves being inferior). Defining f_{1i} to be the number of the rank on which the White King stands and f_{2i} to be the distance from the nearest vertical edge of the board, for class i, will then induce White

to move his King towards the eighth rank if possible and, subject to this, towards the centre files of the board.

Although positions (in the same class) are effectively ranked on the basis of choosing the values of certain functions to be as large as possible, a very common requirement is, for example, to make the distance between the two Kings as small as possible. This can easily be arranged by defining the value of a function to be the negative of the distance between the Kings and using this function in the normal way. (In practice, it is convenient to add a constant to this negative value to ensure that the value of the function always lies in a suitable positive range.)

The notation f_{1i}, f_{2i} etc. will henceforth be abbreviated to f_1, f_2 etc. when no confusion can arise.

2.5 Specifying the associated functions

In general, many of the functions defined for a particular endgame may be associated with more than one class. If there are M functions altogether, then each is given an identifying index number from 1 to M, inclusive. In a practical implementation, index numbers of the functions associated with class i are held in row i of the value table, from column 2 onwards. If for some endgame the largest number of functions associated with any one class is m, then (m+1) columns are required in the value table. For any class with a smaller number of functions, the rightmost (i.e. highest subscripted) columns of the corresponding row of the table are filled with zeroes. These correspond, in fact, to the null function described in Section 2.6.

Taking, for example, the case where m is 3, the entries 1, 4 and 5 in columns 2,3 and 4 respectively of row 10 of the value table may be interpreted as "use function 1 for f_1, function 4 for f_2 and function 5 for f_3 with positions in Class 10", or, more informally, "within Class 10, first choose the largest value of function 1, followed by function 4, followed by function 5".

2.6 Computational considerations - function value and position value

The procedure for making a complete ranking of all successor positions of p in order of favourability can be viewed in terms of sorting the members of Q(p) into order, where the sorting is on the basis of:

(i) class value

(ii) the value of the first associated function (primary function)

(iii) the value of the second associated function (secondary function)

and so on, with the lowest level of sorting arbitrary.

The procedure for finding only the most favourable position (which is, of course, the usual requirement in practice) corresponds to a number of consecutive series of comparisons:

(i) to find the largest class value

(ii) to find the largest value of the first associated function amongst those positions tied at (i)

(iii) to find the largest value of the second associated function amongst those positions tied at (ii)

and so on, as necessary, any residual ties being resolved arbitrarily.

A computationally more efficient method of implementing both these procedures, using no more than one series of comparisons, is as follows: If the associated functions for each equivalence class are specified in such a way that their values are always positive or zero integers less than some positive integer K (where K is chosen to be an upper bound for every class), then the function value of a position $q \in C_i$ is defined as the following weighted sum of the values of the associated functions (evaluated at position q):

$$K^{m-1} \times f_{1i} + K^{m-2} \times f_{2i} + \ldots + f_{mi}$$

where there are m associated functions for class i.

The position value of $q \in C_i$ is defined as the sum:

$$K^m \times \text{class value} + \text{function value.}$$

It should be noted that this value can be decomposed uniquely to give the individual contributions of the class value and each of the associated functions. In practice, it will normally be possible to define all functions in such a way that they take only positive or zero integer values. A suitable value for K will normally (but not always) be ten, since most of the functions chosen will correspond to distances between pieces or between squares on the board.

Providing that the value of m is the same for every class, the ranking of the members of any given set of positions with Black to move is exactly mirrored by the numerical ordering of their corresponding position values (the larger the position value, the more favourable the position to White). Thus the procedure for ranking all the positions in order of favourability is equivalent to simply a sorting of the numerical position values into descending order of magnitude. (The particular algorithm used for sorting will automatically take care of residual ties.)

An algorithm to find only the most favourable successor position (and thus the best move for White) requires only one series of comparisons to find the largest position value, again with the particular algorithm used automatically taking care of residual ties, rather

than up to m + 1 separate series of comparisons (one for the class values and one for each of the associated functions).

Using position values is therefore a convenient and efficient way of implementing both the move-finding and the full ranking algorithms. It should be noted however, that the method has the minor disadvantage of requiring the value of every associated function to be calculated for all the successor positions, not just those involved in ties.

Although the formula given for calculating the position value of q resembles a traditional evaluation function, it should be borne in mind that this is merely a computationally convenient method of representing the ranking process described in Sections 2.2 and 2.4. As noted above, the position value can be uniquely decomposed into its component parts, the class value and the value of each associated function.

Virtually any conceivable method of ranking positions could in principle be converted into some evaluation function (since the integers are an ordered set). The important point is that in this case the component parts of the position value (evaluation function) are meaningful in themselves, the class value corresponding to an ordering of priorities (goals) and the associated functions to geometrical properties of the position, such as distances to be made as large (or small) as possible, in turn.

One important point has been omitted from the above discussion. It has been assumed that all the equivalence classes have the same number of associated functions. In general, however, this will not be the case. This problem can be overcome by the use of a null

function, which has value zero for every position q. If the largest number of functions associated with any class is m then for each class with a smaller number of functions, null functions are appended in the "least significant" positions to bring the total to m. Thus if the maximum number of functions for any class is 3, then if for some class there is only one associated function this is taken as the primary function, with two null functions appended for the two least significant functions.

2.7 Summary of the basic model

This section contains a summary of the method of creating the move-finding algorithm for a given endgame.

For the endgame under consideration, it is necessary to devise a set of rules (predicate functions) for allocating each position with Black to move to one of a number of equivalence classes of positions, together with a fixed ranking between the equivalence classes and a number of functions associated with each class, used to rank positions within that class.

The ordering of the rules should be chosen so as to simplify their definitions wherever possible. The order of the class values corresponds to the relative favourability (from White's viewpoint) of the positions in each class.

A computationally efficient algorithm for finding the best move in any position p with White to move is then as follows:

(a) generate the set of (immediate) successor positions of p;

(b) calculate the value of each successor position using steps

    (b1) to (b4)

(b1) evaluate each of the rules in turn until the first true rule is found, this gives the corresponding equivalence class

(b2) use column 1 of the value table to find the class value

(b3) calculate the function value from the functions associated with the class (using column 2 onwards of the value table)

(b4) calculate the position value (using K, an upper bound for the value of every function, and m, the number of functions - including null functions - associated with each class);

(c) find the successor position with the highest position value;

(d) select the corresponding move for White.

A detailed example of the use of this method in practice, for the end-game King and Rook against King, is given in Section 4.

2.8 The extended model

The basic form of the model, as described in Section 2.2, is in principle of general applicability. As will be seen in the following sections, this form of the model appears to be sufficient to produce algorithms performing at a high standard of play for the elementary endgames discussed in this thesis. As the complexity of the endgame increases, however, it seems likely that a point will be reached where it becomes too complicated in practice to specify the necessary classes for a reasonably high level of play, although it will always be possible to specify some initial form of algorithm even if only a rudimentary one. There are two possible solutions to this problem: firstly to make use of a process of continuous refinement along the lines developed for the two elementary endgames in the remainder of this thesis. For a complex endgame this may result in a large number of equivalence classes but

may still be a computationally efficient alternative.

The second alternative is to extend the basic model to allow for searching below the level of the immediate successor positions. Changing the meaning of "position value" to include a backed-up value where necessary, the discussion in the previous sections remains valid.

The second possibility accords with the observation that complex end-games, which are not regarded as completely understood theoretically, are generally played by means of a combination of pattern recognition and goal-directed analysis, although even for quite complex endgames the amount of analysis required would appear to be reasonably small. A suitable means of extending the model to allow tree-searching, which enables the amount of search to be carefully controlled, is as follows.

In general, the equivalence classes can be divided into three categories:

(i) "positive" classes - those with a value higher than that of the residual class;

(ii) "negative" classes - those with a value lower than that of the residual class;

(iii) the residual class itself.

For complex endgames, the most likely source of difficulty lies in specifying a sufficiently large number of positive or negative classes, with the majority of positions thus falling into the residual class.

There are four possibilities for a given set of successor positions:

(a) at least one belongs to a positive class;

(b) all belong to negative classes;

(c) all belong to negative classes, except one which belongs to the

residual class;

(d) two or more positions belong to the residual class and no position belongs to a positive class.

In cases (a), (b) and (c) the most favourable position can be found statically using position values in the usual way. In case (d), either the positions in the residual class can be taken as terminal and the associated functions used to calculate the position value statically, or an analysis tree can be generated from each of the residual class positions, with the negative class positions rejected altogether. Constructing an analysis tree (either depth-first or breadth-first) in this way has the effect of reducing the amount of search by pruning all branches to positions in negative classes (unless there is no alternative) and defining terminal states of a given set of positions, with the second player to move, as a whole (cases (a), (b) and (c) above). Since a residual class position can, at any stage, be regarded as terminal, with the static position value (calculated using the associated functions) backed up the tree, the amount of analysis performed is subject to close control. For example, the maximum depth of search or even the maximum number of individual positions to be considered might be specified. Any non-terminal set of successor positions remaining at that stage could then be compared statically by means of position values. If required, a heuristic pruning could be applied to further restrict the volume of search, for example by searching from only the three (or so) positions with the highest static position values, in case (d).

In general, this form of the equivalence class model is distinguished from conventional models based on tree-searching in that search is intended to be used in a controlled way only as necessary to supplement the pattern-matching which it is believed is the fundamental

component of the chessplayer's endgame knowledge.

In view of the existing body of theory relating to tree-searching, such an extension of the basic form of the model does not appear to present any significant difficulties.

2.9 Discussion

In this section, a point raised in Section 2.2 is discussed further: a choice must be made between the successors of a position p without any reference to p itself. This point is, in fact, fundamental to the model which has been described. White's move-finding algorithm is seen essentially as a choice between the successor positions to which he may move. The chosen move is the one which leads to the best position. When comparing positions with Black to move there is no logical justification for referring to any common antecedent the positions may have; they must inevitably be compared on their own merits alone. This approach is in accord with the large majority of programs which have been written to play the full game but not with the (relatively small) number of programs written specifically for endgames, as will be seen in Section 3. In practice, concentrating solely on positions with Black to move appears to result in a substantial reduction in the number of patterns which need to be recognised, partly because considerations of whether or not White has improved his position by his move need not be taken into account and partly because applying pattern matching to positions with White to move is equivalent to placing the members of W into equivalence classes and the membership of such equivalence classes would often be better defined in terms

of the most favourable successor position than by the static features of the positions themselves.

The following example is given as an illustration of this latter point. Figure 1 shows a position from the King and Rook against King endgame, in which White to move can force checkmate in two moves. Note that the following conventions have been adopted for diagrams and for recording moves. The figure number will generally be given to the top right of the diagram with the player to move (W or B) on the left. In all cases White's first rank is taken to be at the bottom of the board. In diagrams, the letters K and R are used to denote the White King and the White Rook, respectively. The Black King is represented by k. The notation used for moves is based on the algebraic chess notation and is intended to be self-explanatory for those familiar with that notation. A full description is given in Appendix 3.

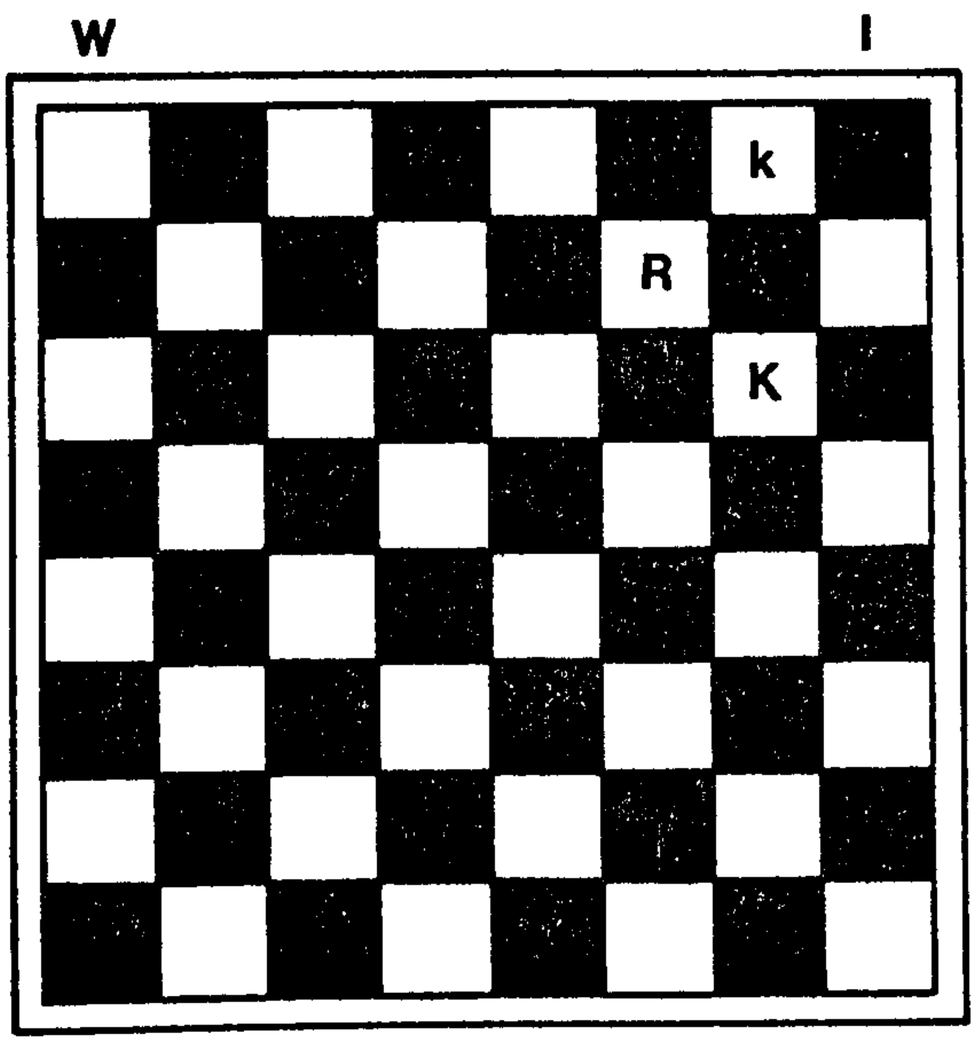


Figure 1

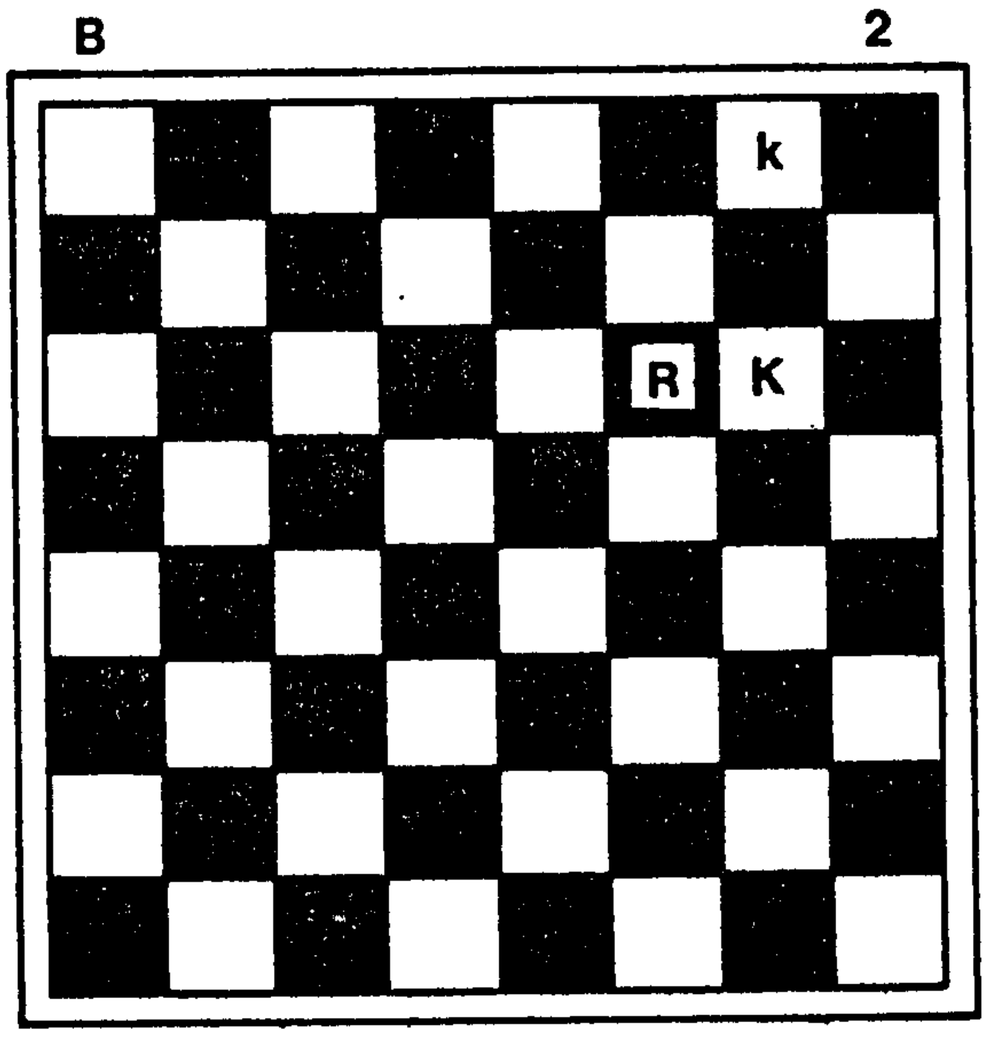


Figure 2

In Figure 1, White wins by 1.R-F6, giving Figure 2. Black must now play 1. ... K-H8, whereupon White checkmates by 2.R-F8.

Now consider Figure 3. White wins by 1.K-G6, again giving Figure 2 with Black to move.

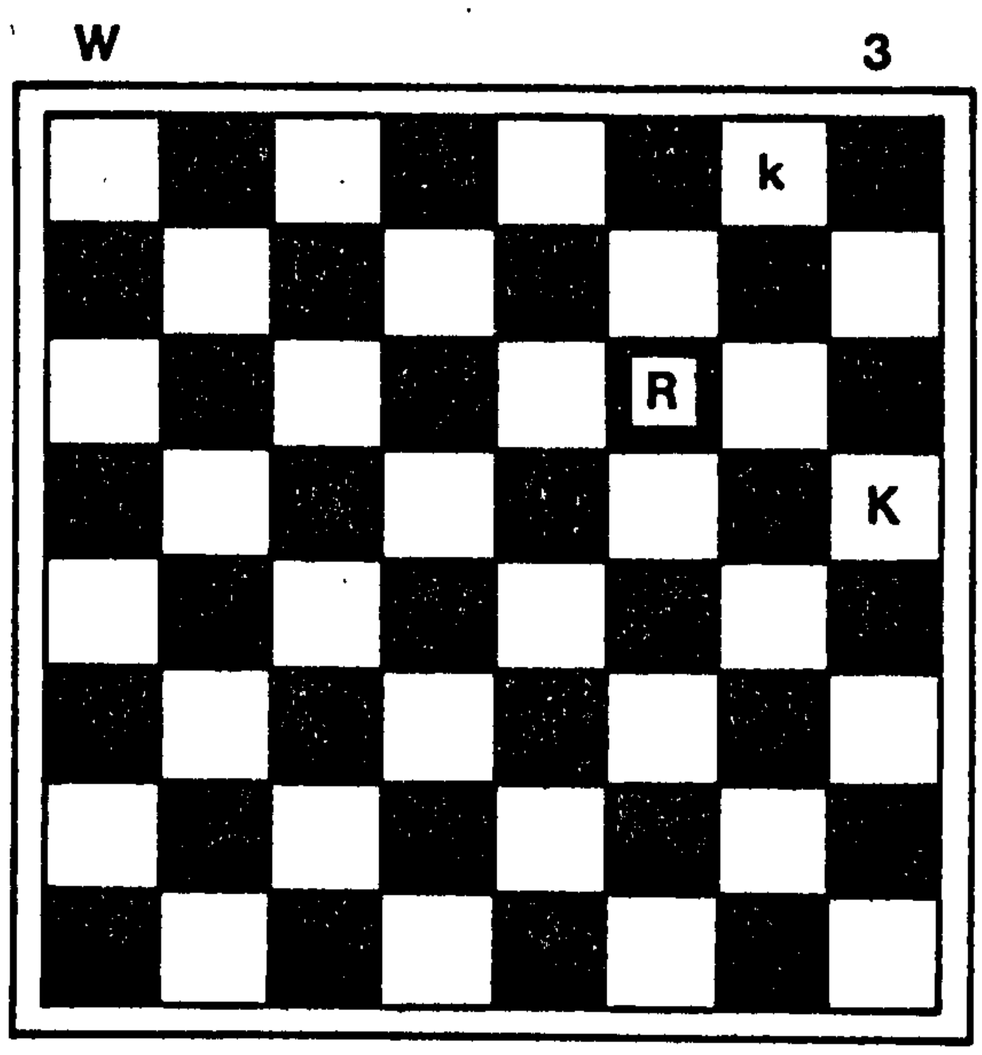


Figure 3

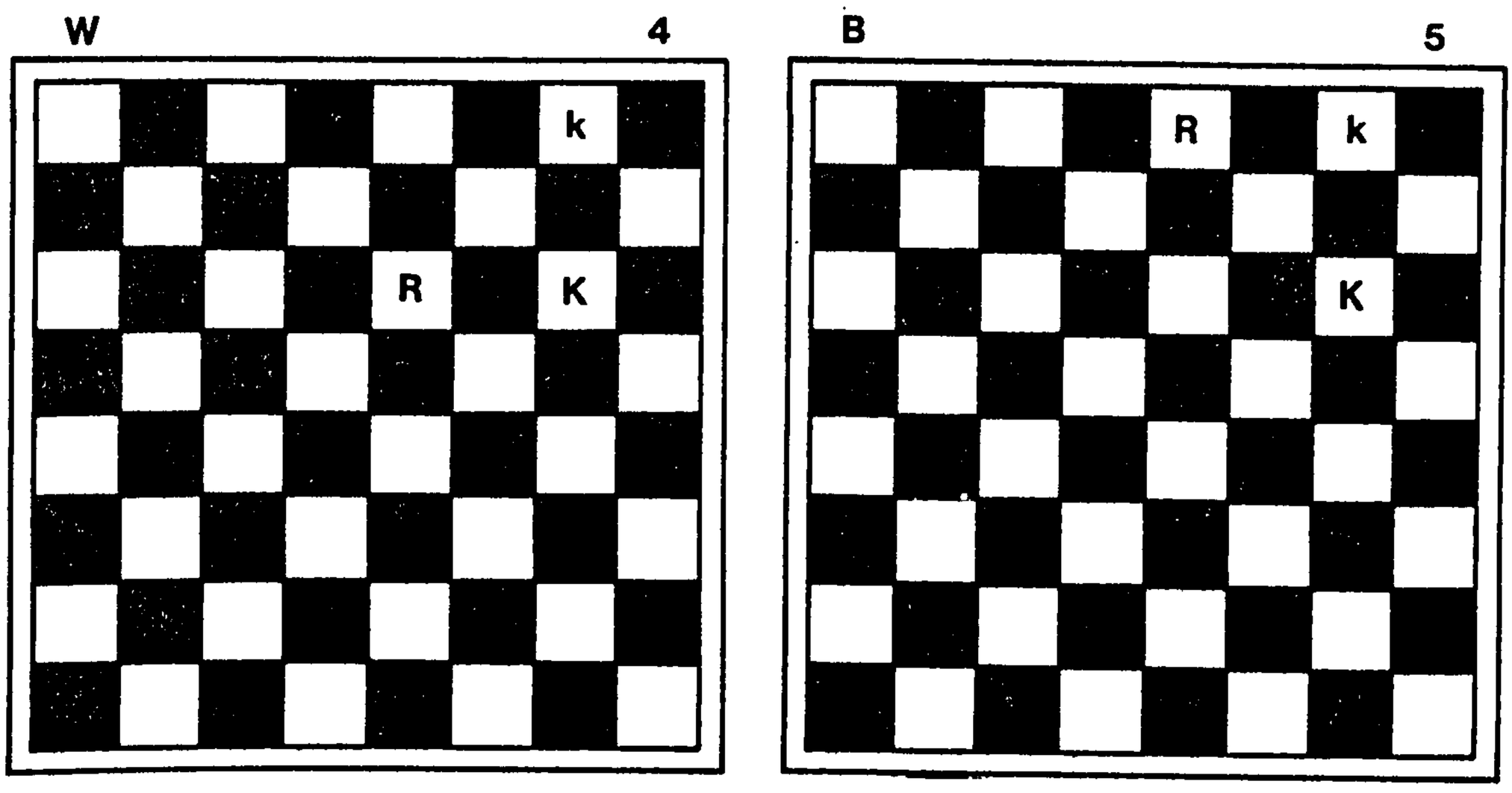
In an intuitively obvious sense, Figures 1 and 3 are equivalent. In each case White's best play is to mate in two moves by first playing

to Figure 2.

There are, in fact, many other positions equivalent to Figure 1 in this sense. For example, with the two Kings as in Figure 1, White's Rook could be on F1, F2, F3, F4 or F5 instead of F7 (the first move being 1.R-F6 in each case). Alternatively, with the Rook initially at F6 the White King could be at H6, F5 or G5 (the first move being 1.K-G6). The number of equivalent positions can be further increased by noting that if White's Rook were on F1, say, in Figure 2, this would make no difference to the outcome. This modified form of Figure 2 will be called Figure 2A (not diagrammed). The positions "White King on G6, Rook on G1" and "White King on F5, Rook on F1" are then also equivalent to Figure 1, since from each one White's best move is to play to Figure 2A (by 1.R-F1 and 1.K-G6, respectively). Proceding in this way, with different modified forms of Figure 2, a very large number of equivalent positions could be found. What is most apparent about all these positions is their lack of similarity in terms of the positions of the pieces. Defining the members of an equivalence class of positions (with White to move) equivalent to Figure 1, by means of the static features of the positions themselves, must inevitably require an extremely large number of separate and unrelated tests, as would implementing logic of the form <u>if</u> <mate-in-two position> <u>then</u> <play first move in sequence>.

The real equivalence between all the positions lies not in their static features at all but in the fact that in each one, White's best move is to play to Figure 2 or a position effectively identical to it, such as Figure 2A.

By concentrating on specifying classes of equivalent successor positions, with Black to move, the problems of definition encountered above become trivial. The equivalence class of positions such that "Black to move cannot avoid mate in one" (of which Figures 2 and 2A are members) can be readily defined and the overall White strategy "select best available position" takes care of all the details.



The above example can be extended to illustrate one further point. In Figure 4 White can play 1.R-F6, once again giving Figure 2. His best move, however, is to give checkmate immediately by 1.R-E8 mate. Figure 5 shows the resulting checkmate position. Thus Figure 4 should not be a member of an equivalence class containing Figure 1 "White can checkmate in two moves" but of a second class "White can checkmate in one move", of which there are, of course, many other members. The first of these classes is then strictly "White can checkmate in two moves but not in one move".

The allocation of a position p (with White to move) to one equivalence class rather than another thus seems to need to be based on the most

favourable successor position to which White can move from p, with a corresponding increase in the complexity of the necessary definitions. Again, this problem can be overcome in a simple way by defining a class of positions with Black to move equivalent to Figure 5, "Black is checkmated", ranking this class higher than that containing Figures 2 and 2A and using the rule "select the highest ranked successor position". In summary, it would appear that a model based on a ranking of equivalence classes of successor positions, with Black to move only, offers a number of significant advantages over one which makes use of pattern matching applied to positions with White to move.

In the following section, previous work in the area of Chess endgames is discussed and compared with the approach described in this thesis.

3. Discussion of previous work

In this section the principal previous work in the area of endgame-playing programs is considered in detail in the context of the objectives set out in Section 1.

Other related work on learning programs, inductive generalization etc will be considered as necessary in Section 11.

3.1 Torres' machine

The first endgame mechanisation did not involve an electronic computer but a special-purpose (electro-mechanical) machine built by Torres y Quevedo in the late nineteenth century to play King and Rook against King, described by Vigneron (1914) and mentioned by Shannon (1950).

The machine's "algorithm" was built entirely into its electrical circuitry. Only moves for the side with the Rook (referred to as White below) were generated. Michie (1976a) gives a reconstruction (making common-sense interpretations where Vigneron's text is unclear) in the form of a decision table with six rules, composed of combinations of eight conditions, and five possible actions. The conditions are simple predicates such as "distance in ranks between White Rook and Black King is greater than one". The actions are of the form "move the Rook so as to maximize the file distance between it and the Black King" or "move the Rook horizontally one file, giving preference to the move away from the Black King".

The machine only operated when the initial position satisfied the pre-condition that the White Rook did not occupy either of the centre files and was between the ranks of the two Kings, with the

Black King on a higher rank than the White. The algorithm is compact and makes no use of tree-searching, however its performance looked at objectively can only be described as very poor. Michie reports that the machine takes as many as 62 moves to checkmate in the worst case, thus exceeding the maximum of 50 moves permitted by the laws of chess and allowing the opponent to claim a draw.

Nevertheless in the cases where the initial position satisfied the precondition, the machine always seems to find a move and ultimately arrive at a checkmate.

Subsequent mechanizations of King and Rook against King have improved the overall quality of White's play but only at the expense of the compactness of the algorithm and the certainty of winning from all possible starting positions or even, in some cases, of always finding a move.

3.2 Huberman's programs

The first computer programs to play chess endgames were written by Huberman (1968), who constructed programs for the three endgames King and Rook against King (KRK), King and two Bishops against King (KBBK), and King, Bishop and Knight against King (KBNK), the principal aim of this work being to study the process of translation from textbook descriptions of endgame play into computer programs. Huberman's work is an example of a structural (as opposed to procedural) form of representation. All three programs are written in LISP and make use of the same overall model and find moves for White(in each case the player with the additional material) only. The model is only intended to be applied to positions in which White has a forced win against any play by Black. If given any other position, it is unlikely that any move would be found. The model used by Huberman is a forcing tree. Each program uses two predicate functions better and worse which take

as arguments an initial position p with White to move and a successor position q (at some depth), with Black to move. A breadth first tree search is used with _better_ positions as goals for White. Any variation which leads to a _worse_ position against best play by Black is immediately rejected, thus no successors of a _worse_ position are ever generated.

A _better_ or _worse_ position is any position with Black to move for which the function _better_ or _worse_ is _true_. These two functions are in no respect complementary; _better_ is used to define goals for White and _worse_ is used to avoid disastrous continuations such as stalemate. Both functions are also used for purposes of pruning the analysis tree. In the resulting "forcing tree", there is one branch from each position with White to move. For positions with Black to move, there is one branch for each of Black's legal moves. The terminal positions are all ones in which Black is to move and the predicate _better_ is true.

Once a forcing tree has been constructed, all White's moves are taken from the tree until a terminal position (i.e. a _better_ position) is reached. For White's next move a further forcing tree is constructed using the current position as p, when evaluating _better_ and _worse_, and with no memory of the previous tree. The intention is that the predicates should be defined in such a way that White can always force a _better_ position q and that, in some sense, q is closer to checkmate than p so that the process ultimately converges.

To implement _better_ and _worse_ Huberman defines a number of _stages_ for each endgame, each corresponding to one book heuristic. Each stage contains positions with both Black and White to move and in most cases there is a numerical _measure_ (e.g. the distance between the

two Kings) associated with each stage.

In general, the "higher" the stage and the smaller the value of the measure the more favourable the position to White. Function <u>better</u> is, roughly speaking, defined by the criterion that q should be in a higher stage than p or in the same stage with a smaller value of the measure. Function <u>worse</u> is defined essentially by the criterion that q is a member of stage 0. Clearly a simple three-stage program, consisting of positions such that 'Black is stalemated or can immediately capture a piece', 'Black is checkmated' and 'all positions not otherwise included' would perform perfectly, but at the expense of a prohibitively large amount of tree-searching. Introducing additional stages can therefore be seen as a means of reducing the amount of searching required. Since Huberman's aim is that each stage should represent one textbook heuristic, the major burden of reducing the depth and breadth of the necessary search falls on fairly <u>ad hoc</u> extensions to <u>better</u> and <u>worse</u>, respectively. Other steps taken to reduce the quantity of searching required include a weakening of the definition of <u>better</u> given above (to allow q to be in the same stage as p with the same value of the measure) a "redundant branch cutoff" heuristic and a careful ordering of move generation. In fact, since the programs consider all <u>better</u> positions to be equally good and accept the first <u>better</u> position generated at any level, the order of move generation is a critical part of the algorithm.

Huberman gives definitions of <u>better</u> and <u>worse</u> for the three end-games previously mentioned. For KRK there are 4 stages (only 1 with a measure), for KBBK there are 5 stages (3 with measures) and for KBNK, 7 stages (2 with measures).

However,the definitions of the stages and the necessary extensions

to better and worse to reduce the depth and breadth of search to manageable proportions are in each case extremely complex combinations of predicates with little overt resemblance to the book heuristics remaining. Huberman comments "It is simple to decide roughly what the stages are, and what kind of heuristic each requires. This information is often stated in the books. It is difficult to give the exact definition of the stages and measures, and generally it is even more difficult to define the additions to better and worse which make them practical".

Positions taken from Fine (1944) and Capablanca (1935) are used as starting positions for a small number of examples to illustrate program play. The play for KRK seems to be strong in the cases given, the play for the two more difficult endgames is less strong but still sufficient to win and is particularly effective in the final stages of the examples given. Program play is not optimal for any of the three endgames and this is not an objective of Huberman's work. In one case a checkmating move (in KRK) is not played, a consequence of the principle that all better positions are equally good and of the particular order of move generation.

Unfortunately, no indication is given of the extent of any empirical testing which may have been performed other than the few examples of program play quoted. It is therefore impossible to judge the quality of program performance in general without access to the programs themselves.

Huberman states that her programs can be proved to be correct, in the sense that White eventually wins against any defence by Black. However, such proofs do not indicate the level of search required for White to force Black from one stage to the next. It would also

seem to be extremely difficult to ensure correctness for any more complex endgames.

In the examples given a maximum depth of analysis of 7-ply is required for KRK, 15-ply for KBBK and 13-ply for KBNK, in examples of 17, 21 and 40 (White) moves, respectively. Despite the importance of Huberman's work as a structural and reasonably generalisable approach to programming the endgame, and her achievement in creating programs for three endgames, at least one of which is difficult by human standards (KBNK), there are a number of inherent weaknesses in her approach.

Most significantly, the method is only applicable to starting positions where White has a forced win and the searching process will not terminate otherwise. Unfortunately, it is difficult to specify precisely the subset of positions in which White has a forced win, even in the case of KBNK. A second difficulty arises from the necessity to treat all <u>better</u> positions as equal,which tends to produce inferior play and thus inevitably prolong the game. Compensating for this leads to more complex definitions of stages and the function <u>better</u>. There is little control over the depth and breadth of search required and, in fact, a very careful definition of stages and the extensions to <u>better</u> and <u>worse</u> is required to ensure that the program always returns a move (whether or not the program is correct). Refinements to the program make this condition progressively more difficult to ensure. In addition the condition that <u>better</u> and <u>worse</u> are always defined by comparison between a position q with Black to move and the position p at the root of the tree seems somewhat restrictive. The requirement that each stage should contain positions with both White and Black to move also leads to complications in some cases, and in some instances to different definitions with White and Black to move.

Huberman's work has been described in detail here because it bears certain similarities to the equivalence class model described in this thesis. In particular Huberman's stages and measures are similar to equivalence classes and associated functions. From the viewpoint of the equivalence class model, the principle differences are as follows.

(i) Classes are defined with only one side to move;

(ii) Each class can have more than one associated function.

(iii) The only comparison is between positions with Black to move, and the most favourable position (from White's viewpoint) is chosen.

(iv) A move is found from every initial position.

(v) No tree-searching has been found to be necessary for the two endgames discussed in this thesis. For more complex endgames it can be used in a controlled way, since at any point there is always the alternative of accepting a member of the "residual class" as a terminal position and making use of the associated functions.

(vi) Refinements to the algorithm cannot affect the property that a move is always found from every initial position.

(vii) There is no a priori means of ensuring that the program will always win from positions where that is possible.

3.3 Tan's Program

Programs for the endgames King and Pawn against King (KPK) and King and Pawn against King and Bishop have been written by Tan (1972 and 1974, respectively). The second program is not fully reported and only the first will be considered here.

The principal aim of the work is to illustrate a particular way in which "knowledge might be represented, organized and used". The

program's knowledge is embedded directly in POP-2 procedures.

Moves are found for both sides, making the simplification that the game is over and won for White (always assumed to be the side with the Pawn) if the Pawn promotes and cannot immediately be captured. The 50-move and three-fold repetition of position drawing rules are not implemented and the program does not explicitly avoid repetitions of position.

The overall organization of the program consists of an ordered sequence of conditional statements of the form: if condition then action scheme, organized to form a decision tree. Tan states that whereas the content of these statements can be obtained from textbooks, their ordering cannot and that "differences of strength of players ... are mainly due to organizational structure of their chess knowledge".

In some cases action scheme includes the "value" of the resulting position (win for White or draw). Where it does not, the program will automatically play on, finding the "best" move for each side alternately until a position of known value is reached. This facility enables a simple linear plan to be stored corresponding to play considered best for both sides. In subsequent play, the program uses this plan to find its moves, as long as the opponent does not deviate.

The decision tree has six main branches, depending on which side is to move and whether the Pawn is on the sixth rank, the seventh rank, or another rank. The main branches divide into sub-branches depending on the value of predicates such as "Rookpawn" and "the Pawn can advance without immediate capture". For each sub-branch, conditions are tested in order until one is satisfied, at which point a

corresponding action scheme is used to generate a move. Although some of the conditions are simply predicate functions applied to the initial position, others are considerably more complicated. Thus, one condition is effectively "can the player to move take up the following pattern of pieces by a King move?" Another requires "trying" the specified action, i.e. a move is found and the resulting hypothetical position is then considered for the other side. An action scheme is found for that side to move which may have a value (win or draw) associated. If not, the value is found by further analysis, as previously described. This value is then considered in the context of the original position. If it is a win, in the case of a position with White to move, or a draw, in the case of a position with Black to move, the "tried" action is accepted, otherwise the next conditional statement is considered, as usual.

Since trying an action is included in the strategy for both sides, this type of condition may involve a depth-first tree search. There is no explicit control on the size of this tree and in a situation where the program played in such a way that the same cycle of positions was repeated continuously, without a win or draw value being assigned, the program would fail to terminate, and thus no decision on the action tried in the original position would be made. In general, the size of tree is only likely to be manageable if the set of positions for which known values are given is as large as possible. The method of "trying an action scheme" is only applicable in the case of a program which finds moves for both sides and where the values of positions are known or can be calculated reasonably simply.

A final kind of condition, which only occurs with White to move, is to test whether the White King can legally move to any one of an ordered list of "critical squares" which depend on the position of

the Pawn. If it can, the first such move in order is taken.

There are eleven elementary conditions (defined by a combination of predicate functions or by pattern matching) applied to positions with White to move and 11 applied to positions with Black to move. The six main branches divide into a total of 19 sub-branches.

There are five elementary types of action scheme used by the program: to move either King to a specified square, to advance the Pawn either one or two squares (with White to move) or to make no move (used when either player is stalemated). One further action scheme, for each side, is to move the King as close as possible to the White Pawn. Three further action schemes defined with White to move involve a series of decisions used to select one of the elementary actions. In addition there are five "manoeuvres", 2 for White and three for Black, which involve using combinations of predicate functions and pattern matching to select an elementary action. For one manoeuvre, with White to move, the condition "can White move his King to take up the following position?" is used six times, together with other tests. The most complicated action scheme consists of "trying" a succession of King moves in turn, in a way similar to the condition of "trying an action scheme" described previously. King moves are sorted so that those in the direction of "critical squares" (depending on the position of the Pawn) are tried first. The first move found which results in a win (if White is to move) or a draw (if Black is to move) is accepted. This type of action scheme can occur with either side to move and can again result in depth-first tree searching not subject to direct control.

In cases where White to move cannot win, or Black to move cannot draw, all King moves will be examined by the program. A total of

eleven action schemes (plus lookahead) are used in the decision tree with White to move, and twelve (plus lookahead) with Black to move.

Tan states that his program solves all known cases, including all examples from Awerbach (1958) and Fine (1941) and that the program is believed to be correct or almost correct, in the sense of preserving the value (win or draw) of a position. This condition is not, of course, sufficient to guarantee that White wins wherever possible since, for example, cycling may occur. As pointed out previously, in such a case it is possible that in certain positions a move will not be found and the program will fail to terminate.

Only one simple example is given where White wins in six moves using a four-move lookahead.

Although Tan's program is applicable to all positions (in contrast to Huberman's where a winning position is required), the tree searching required makes it essential to know the values of as many non-terminal positions as possible (to reduce the amount of searching) and to construct the algorithm in such a way that the value of a position is always preserved and cycling never occurs (or the algorithm may not terminate in some positions). For more complicated endgames both these conditions may prove extremely difficult to satisfy. When playing White, the program accepts the first branch which leads to a win (just as Huberman accepts the first better position), although there may be better moves available. This leads inevitably to inferior play, which can only be improved at the expense of a further increase in complexity.

Since the form of Tan's decision tree is dependent on the domain to which it is applied, his program is an example of a purely procedural representation.

However well the program may play, the algorithm cannot be applied (except in the most general way) to any other endgames and is far too complex to make learning new conditions or action schemes a feasible possibility. These were not, however, goals of the research.

3.4 Zuidema's program

Two versions of a program for King and Rook against King (KRK) have been described by Zuidema (1974), an international chess master. King and Rook against King is used as a means of investigating the problems involved in programming the full game of chess, on a convenient scale.

The traditional tree-searching and evaluation function model is used with a depth of only one ply, i.e. the immediate successors of an initial position are considered when finding a move. The problem of particular concern is whether it is possible to convert chess heuristics and non-material considerations into numerical evaluation functions. Each program plays a complete game as White (always assumed to be the side with the Rook) against a human opponent, from any legal starting position. Repetitions of position by the program are explicitly prevented by recording a "history" of positions which have previously occurred with Black to move. The fifty-move drawing rule is not implemented, although the number of moves played is available throughout the game.

The second version of the program is a refined form of the first, within the same overall framework, intended not only to remedy some of the weaknesses found in the play of the original version but also as a means of investigating the programming effort required to correct "exceptions".

The overall form of the algorithm used to find a move for White in the first program is as follows.

(i) The position is transformed to a standard form (essentially one where the Black King is in the triangle E5-E8-H8) by a suitable combination of reflections about axes of symmetry.

(ii) A checkmate in one move or, in some cases, a checkmate in two or three moves is recognized directly and the corresponding move made. To recognize these the program uses the function measure, defined as the sum of the squares of the rank and file differences between the two Kings.

(iii) If no move is found at stage (ii), and measure > 15 then a White King move towards the Black King is made immediately, unless the Rook is en prise.

(iv) If no move is found at stage (iii), certain moves are immediately rejected without further investigation:

(a) Rook moves to the third file or the third or fourth rank;

(b) Rook moves to the second file (or rank), unless it was previously on the first file (or rank).

(v) The remaining moves are generated in turn in a prescribed order and the value of a function room is calculated, for each corresponding successor position, together with the new value of measure in the case of a King move. Any move which leads to a position in the "history" is automatically rejected, as is any move which gives Room = 1 (i.e. a stalemate position), or which increases the value of measure by more than 1. For the remaining moves, the values of the functions room and measure for the successor positions are combined into

an evaluation function in such a way that the position with the smallest value of room, and subject to that the smallest value of measure, is found and the corresponding move is played. In the event of a tie the first generated move is taken.

The value of room is essentially the number of squares in the quadrant to which the Black King is confined by the Rook, with four principal adjustments to allow for the position of the White King and others to allow for situations such as those where the Black King is closer to the Rook than the White King or where Black is in check and can move into a larger area of the board.

Zuidema states that this version of the program will always checkmate within the fifty moves permitted by the rules, however no indication is given of the extent to which the program has been tested. One of the two examples of complete play given shows a checkmate in 27 moves (i.e. 53 ply). Michie (1976a) points out that the theoretical number of moves required to win from the initial position given is only 14 (27 ply). To improve the performance of the program, the following three changes are made in the second version.

(a) The order in which King moves are generated is modified.

(b) A further function distance is introduced, defined as the sum of the squares of the rank and file distances between the Black King and the Rook. The definition of the evaluation function described in (v) above is changed so that any tie amongst successor positions, after taking the smallest values of room and measure, is resolved by choosing the position with the largest value of distance. Only if there is still a tie will the order of move generation be significant.

(c) The definition of room is considerably more complex. There are now fifteen possible types of adjustment to the original definition, depending on the result of a series of tests. Some of these types of adjustment themselves require further series of tests to be applied to decide on the numerical adjustment to make. In terms of lines of ALGOL 60 program text, the definition of room is increased from 45 lines to 135.

The overall improvement gained by these modifications is not easy to judge from Zuidema's account, although several examples of improved play in specific positions are given. However Zuidema gives other examples of positions where poor moves are still played. Correcting these would involve a further increase in complexity.

On the basis of this work, Zuidema's conclusions on the prospects for programming the full game of chess are generally pessimistic.

For his first program he states "Level of play is not high. For any rule given in the program one is able to construct exceptional situations. A more refined strategy is needed to elevate level of play. A small improvement, however, entails great deal of expense in programming effort and program length. The rules will have their exceptions too. Exceptions that will not even be noticed by human players."

On the subject of the refinements to the first version of the algorithm, he states that:

"A small improvement entails a great deal of effort. The conclusion forces itself that refining the algorithm and exceptions to rules give rise to an overburdened program."

Zuidema's work is less readily classifiable than either Huberman's or Tan's, however as the use of functions room, measure and distance is unlikely to be applicable to more than a few other endgames, it is possibly best described as procedural.

The use of these three functions by Zuidema is similar in some respects to the use of associated functions in the model described in this thesis. From the point of view of the equivalence class model, the differences are that the predicates defining the different types of successor position are explicitly stored (as the rules defining equivalence classes), rather than used to determine the value of a single function (room). This enables different functions to be associated with each class, if required. The order of move generation is intended to be used as a means of making an arbitrary choice between positions, whereas in Zuidema's programs the order would seem to play an important part in the selection process. Zuidema describes briefly a further program to play King and Queen against King, which also uses the idea of room, with only one ply tree search. He suggests that a similar method could be applied to the endgames King and two Bishops against King, and King, Bishop and Knight against King, in both cases with a deeper tree search.

It is not clear, however, how the rather ad hoc methods of pruning White moves initially and rejecting successor positions subsequently would carry over to such an extension to deeper searches.

Certain features of Zuidema's method would also seem to make it unlikely to be readily applicable to other endgames. Firstly the use of only one set of functions (such as room, measure and distance) for any endgame is likely to require the definition of some of the functions to become excessively complex. Secondly, it is improbable that any

order of move generation will be entirely satisfactory for a given endgame, particularly because of the effect of possible reflections of a position about an axis of symmetry. Compensating for a poor order will involve an increase in complexity of function definition and possibly an increase in the amount of tree-searching required. Finally, Zuidema's use of "history" to avoid repeating a (successor) position and of initial "pruning" of positions with White to move is considerably more significant than simply avoiding draws by repetition of position and reducing the number of moves to be considered. In fact, both of these are critical parts of the algorithm. Examples are given which show a good move being selected only because an inferior move, which leads to a repetition of position, is rejected. Without the pruning of positions with White to move and the recognition of checkmating positions etc., a further substantial increase in the complexity of the definition of room would be necessary and the original definition as the number of squares in the Rook's quadrant would almost certainly be completely lost.

Unlike Huberman's and Tan's programs, Zuidema's programs do not make use of any tree-searching (below the level of the immediate successors) in the form given and a move will always be found in any position regardless of the quality of the algorithm itself, except in the unlikely event that all moves are rejected entirely.

3.5 Michie's programs

Michie (1976a and b) describes two approaches to programming the endgame King and Rook against King (KRK), the second being a completely revised and improved version of the first. His work is aimed at developing minimal - path algorithms using a structural approach to embedding domain-specific endgame knowledge. For Michie, programming the game of chess is not an end in itself (as it would appear to be for Zuidema); he is concerned with developing a general algorithm to

make use of domain-specific bodies of knowledge.

The principal requirements for a model are that the program modules and data elements should be few and simple and that the process of building and refining the strategy should be straightforward and not burdensome. A representation of knowledge in the form of patterns is considered, but the problem of modifying such patterns in the light of experience is not discussed.

The model developed in the former paper comprises a database of patterns together with a table of advice triples each with the three components: "pointer to condition pattern", "pattern specifying plausibility class" and "pointer to goal pattern (s)". For King and Rook against King, each pattern consists of a specification of the lowest and highest values permissible for the horizontal and vertical co-ordinates of each piece. A pattern thus corresponds to many positions, defined by all permissible combinations of the co-ordinates of the three pieces. The positions of the White King and Rook (White is assumed to have the Rook throughout) are given relative to the Black King. A co-ordinate may be left as "undefined" to indicate any (specific) value. The method of constructing suitable patterns from a given set of positions is not described.

To find a move in a given position, it is first matched against each condition pattern in turn until a match is found. Legal moves permitted by the corresponding "plausibility class" are then made in a prescribed order,matching the result in each case against the goal pattern or patterns specified by the advice triple. As soon as a match is found, the corresponding move is output. A "plausibility class" is simply a list of binary digits indicating whether or not each possible move of the King or the Rook is permissible.

From the description given it would seem that the advice table must be set up in such a way that some "condition-match" will always be found and such that some legal move permitted by the plausibility class will then always lead to one of the prescribed goal patterns. The order of legal move generation is fixed over all the plausibility classes and thus would seem to be an important part of the algorithm, as does the ordering of the advice triples in the table. Three examples of the use of this model are given and in the first two cases strategies for both White and Black are implemented. The first example concerns the problem of checkmating (in cases where this is possible only) on a board which is infinite but for a single edge. Eleven patterns are defined for this example, together with a table of eleven advice triples (8 with White to move and 3 with Black to move). For the second example, of checkmate on an infinite board with a corner (from which checkmate can always be forced), with some fairly minor restrictions on the starting position, a pattern-base of 28 patterns is required, with 36 advice triples (21 with White to move, 15 with Black to move). The third example is an implementation of Torres' algorithm (see Section 3.1), for White only, which uses 8 patterns and 6 advice triples (one of which has as many as four goal patterns). The strategies are implemented in POP-2 and informal mathematical arguments are used to demonstrate that the first two are optimal.

In the second and third examples, however, there is an important extension to the facilities available in specifying a pattern. Instead of using an interval to specify unconditionally the smallest and largest possible values of a co-ordinate, a POP-2 predicate function may be used, for example "is piece in same zone as BK?".

For the second and third strategies implemented, three and six such

function calls are required, respectively.

Whatever the merits of the strategies given in the paper, the limited formalism available for specifying patterns by means of co-ordinate intervals and the restriction of a fixed move order throughout all plausibility classes suggests that the model in the form given is unlikely to be applicable to other endgames or perhaps even to an optimal strategy for King and Rook against King. The need to add the use of predicate functions to the original form of the model would seem to support this view. A serious additional limitation is that it is not clear how the searching, which would inevitably be required for a more complex endgame, would be incorporated into the model.

Michie states that whereas implementing Torres' strategy was itself easy, "the limitation of descriptive form to conjunctions of predicates defined on individual co-ordinates was felt as distinctly cramping". He concludes that "more complex domains than KRK will require less restrictive formats" and that "more difficult problems - even KRK on the 8x8 board - will need means of automatically generating patterns", to remove the burden (and likely error) involved in constructing patterns and to enable the formal proof of a strategy's optimality to become a tractable problem.

The second paper is a general report on the development and use of a POP-2 package known as AL1 ("Advice Language 1") by a graduate class at the University of Illinois,under Michie's direction. The package is a generalised and greatly extended form of the model described in the previous paper. The overall object of the package is "to facilitate the transfer of specialist knowledge about chess endgames into machine memory". However, the package is designed to be applicable to a variety of domains apart from chess. The package

comprises two main modules: an advice module, which takes a position and returns an advice list, and a search module which takes an advice list and returns a forcing tree strategy (in Huberman's sense of the term) to secure goals specified in the advice list. Only the move generation required in the second module is directly specific to chess.

The advice module consists, in principle, of many different advice tables, corresponding to a subdivision of the task domain into sub-domains, with a master table used to select the advice table appropriate to the input position. Thus a strategy for a particular end-game might require the use of one or more advice tables.

The package is designed to enable a user to input his specialized knowledge of a particular domain in the form of advice tables, in as simple a way as possible, without any knowledge of computer programming. Thus emphasis is placed on the simplicity, transparency and ease of modification of the knowledge structures used. Each advice table comprises a series of rules (in the form of a decision table), each rule consisting of a combination of predicate functions. For a given input position each rule is scanned in turn, until the first match is found. The result of this match is not an action but an advice list, corresponding to one or more pieces of advice. Each piece of advice specifies "better goals", "holding goals", "move constraints" and a "depth-bound". The search module is then used to construct a forcing tree strategy to achieve a "better goal" (similar to a better position in Huberman's model), whilst at all times satisfying the "holding goals" (similar to avoiding worse positions in Huberman's terms). The tree of variations is further pruned by the use of move constraints and a fixed maximum depth for each piece of advice. Thus the means of controlling the size of the search tree

is considerably more simple in structure than in Huberman's model. Having selected a rule, the program tries to generate a forcing tree to satisfy the requirements of each piece of advice in the advice list in turn, until a tree is successfully constructed. It is intended that the advice table should be written in such a way that constructing such a tree should always be possible for the first rule satisfied.

A special type of "piece of advice" uses an "empty" better goal, taking as a "default" being at the full depth-bound distance from the root node". This type of advice is used to ensure that the holding goal can be maintained for a specified number of moves. In this case, only the first move in the forcing tree is used in play, with further moves after the opponent's reply found from the table as before. In all other cases, the moves held in the forcing tree are used for as long as possible.

Although Michie's model is a general one, the work described is of an exploratory nature and was not completed at the end of the available period for the graduate class. Thus it is impossible to say at present how effective the model will prove in practice or to judge the number and complexity of the rules and "pieces of advice" required and the amount of searching necessary for endgames of varying degrees of complexity. These considerations would be particularly important in conjunction with the aim of optimal algorithms expressed in the former paper.

An advice table is given for the endgame King and Rook against King, consisting of 10 rules formed from combinations of seven predicate functions. The ten advice lists refer to a total of 30 pieces of advice (24 different pieces), with between 1 and 5 pieces of advice in each advice list.

The table would seem to be for the side with the Rook only and details of four of the predicate functions are given, for example "the Rook does not divide horizontally the two Kings". No details of the individual pieces of advice are given. It is not clear from the description whether the table represents a complete strategy for the side with the Rook, or a strategy for all positions satisfying some pre-condition. The table is stated to be "error free" for a uniform depth bound of two ply. This would seem to indicate that a forcing tree is always generated successfully, rather than indicating the quality of the program's performance. The program is, however, said to produce good play although the level of testing to which it has been subjected is not given. Significantly, the table is said to represent only two student - days of work, which is a good indication of the potential value of such a structural approach. Further tables for King and Pawn against King and Rook (Pawn on the seventh rank only), King and Queen against King and Rook, and King and Knight against King and Rook were constructed by the graduate class. However, these were not fully completed or tested by the end of the available period and no details are given. Work on automatic correction of a table in which an incorrect rule had been introduced is briefly mentioned, but it is pointed out that this form of "concept learning" is relatively unimportant compared with the automatic optimisation of the advice parts of rules. This latter problem was not itself addressed.

Clearly Michie's approach in this paper has a number of features in common with the equivalence class model described in this thesis. In particular both are structural models, designed to be of general applicability, and both make use of an ordered collection of rules (combinations of predicate functions) to classify positions. There are however considerable differences in the details of the two models.

In view of the very general nature of Michie's advice lists, his model is probably of potentially wider applicability. However, the practical effectiveness of an advice table approach cannot be determined at the present time. The greater complexity involved in specifying better goals, holding goals, move constraints and depth bounds may prove a disadvantage from the viewpoint of modifying the algorithm in the light of its experience in play, particularly so for automatic program modification.

The requirement that whenever a rule is satisfied, a forcing tree can always be constructed for one of the corresponding pieces of advice may prove a severe constraint on the building of advice tables for more complex endgames than King and Rook against King. The use of a forcing tree to search for a better position rather than the best position available can lead to serious program inefficiencies, as discussed in connection with Huberman's work above. In cases where the time saved by looking up intermediate moves in a forcing tree, rather than finding them using the advice table, is not significantly large, it may be better to find the best position to the available depth, perhaps by combining the definition of a "better goal" with the use of the equivalent of "associated functions". Retrieving moves from a forcing tree also raises the important theoretical problem of consistency, since it would be difficult to ensure that the same moves would be made if the advice table were used directly. This consideration is of particular importance if the aim is to implement an optimal strategy for any starting position, however no reference is made to the issue of consistency in the paper. A particularly significant difference between the advice table model and the equivalence class model described in this thesis is that the former uses the properties of an initial position as its primary basis for classification of positions, whereas the latter uses the properties of

successor positions (the reasons for this choice were given in Section 2.9 above).

Neither of the two models could readily be modified to allow a combination of these two approaches without changing its overall structure (although Huberman's, Tan's and Zuidema's do so).

Which decision is the more appropriate, as well as the other points raised above, can only be resolved by further experimentation.

3.6 Conclusions

The principal previous work in the area of chess endgame playing programs, together with Torres' machine and the contemporary and ongoing work of Michie, has been described above and discussed in detail. In each case there is little information available on the extent to which the algorithm has been tested and a detailed evaluation of their relative performances is therefore not possible.

The present work, however, is principally concerned with three particular aspects of the algorithms described.

(i) How closely do the algorithms correspond to the chessplayers understanding of the endgame?

Only for Michie is this a major problem. His model would seem to be the only one which, at least potentially, is capable of taking "advice" from chessplayers and being applied as a general framework for a range of different endgames. Huberman's model is also intended to be generally applicable, but the resulting algorithms are extremely complex and not amenable to an advice-giving approach.

All of the algorithms given are complex in relation to the descriptions and examples given in standard textbooks. In general, textbooks make little or no mention of the need to perform detailed analysis for elementary endgames and give only a small number of explicit general rules, with further rules and exceptions embedded in examples. To enable elementary endgames to be represented in the same way by means of a small number of simple patterns with little or no tree-searching, the choice of model is critically important.

(ii) How easily can the algorithms be refined to remove weaknesses and errors in their play?

This problem has not been considered in any detail, although Michie gives ease of algorithm modification as one of the design aims of his model. Zuidema considers the difficulties involved in the refinement process as a serious obstacle, but does not relate this to the particular choice of model.

(iii) How easily could the algorithm's performance be refined automatically?

The only attempt at a self-modifying system was made by Huberman as an extension of her general model. Her method is not described in detail but would seem to require a great deal of preliminary knowledge being given to the program and is probably not a practical possibility in general.

In the following section, an algorithm for the King and Rook against King endgame based on the equivalence class model given in Section 2 is described in detail.

4. An algorithm for the endgame King and Rook against King

The endgame with King and Rook against King is one of the basic endgames given in chess textbooks (e.g. Golombek (1954), Fine (1941)). It is considered elementary by experienced players in the sense that after only a small amount of study, White (the side with the Rook) will invariably win without any difficulty against any defence by Black. The amount of space devoted to the endgame in a textbook will typically be only one or two pages, including examples. Most textbooks, however, omit the ending altogether. All positions with White to move are a win as are virtually all positions with Black to move, the only exceptions being when Black is already stalemated or can immediately capture the Rook. White to move can always prevent these situations arising. Clarke (1975) has calculated that, with White to move initially, the number of moves needed to checkmate against best play by Black is only 16 (i.e. 31 plies) from the least favourable position. Finding this shortest win from every position is a difficult problem but, of course, an unnecessary one for the practical player. The small number of rules given in textbook descriptions appears to suffice for near-optimal play, although no attempt made to cater exhaustively for all possible situations which can occur.

In view of the above it is perhaps not surprising that the ending is never played out in practice (except between beginners), Black invariably resigning before it is reached, with the result that there have been no practical examples in master play (nor probably in competitive amateur play) for hundreds of years.

What may seem surprising, is that constructing an algorithm to solve such an apparently elementary problem should present any difficulties. In fact, however, as the discussion in Section 3 shows, the problem has proved very difficult to solve in a satisfactory manner.

In this section an algorithm for the King and Rook against King endgame is specified, using the model described and discussed in Section 2.

King and Rook against King has been selected as an example principally because it is a relatively simple endgame, thus enabling the reader to make his own evaluation of the effectiveness of the algorithm.

Since the stronger player has a forced win from every position in which it is his own move initially, no awkward problems of determining what constitutes a good move or the best move in a "drawn" position can arise, as they may for other endgames.

The side with the Rook is assumed to be White throughout and the algorithm finds moves for White only.

The algorithm has been derived from an analysis of the information given in a number of standard textbooks, together with some initial empirical testing and refinements.

The principal aim for White (a checkmate position) and the types of position which must be avoided (stalemate positions and those in which the Rook is en prise) are well known and textbooks indicate (by verbal description and examples) the general check-mating strategy which White should follow.

In interpreting this information with sufficient accuracy to incorporate it into a computer program, it is clear that a number of errors - major or minor - will occur. The process of finding and correcting these errors is essentially an iterative one. The form of the algorithm described in this section represents a relatively advanced stage in this process. It appeared to perform well in pilot testing, but no large scale systematic testing had been applied. An example of the initial stages of the analytical process for a more complex endgame is given in Appendix 2.

The algorithm given in this section has been written primarily as an example of how the model described in Section 2 works out in practice, and in this context any considerations of program "optimality" or "correctness" in every possible situation are of secondary importance. The primary concern is the means by which information given in standard textbooks can be embodied in an algorithm. Just as different textbooks and different chessplayers may employ a variety of different strategies for White, so there are many different but nevertheless valid algorithms which may be written using the basic underlying model. It is largely a matter of subjective judgement which of these algorithms is the best. The problem of testing the correctness of any particular algorithm is a complex one, which will be discussed in Section 5. A non-technical description of the algorithm, suitable for evaluation by chessplayers without any specialist knowledge, is given in Appendix 1.

4.1 Preliminary details

Based on an analysis of the rules and examples given in a number of textbooks, the set of legal positions with Black to move for this endgame has been partitioned into eleven distinct equivalence classes. The rules defining these classes and their corresponding class values are given in Section 4.2, followed by a detailed discussion of the rationale for their choice and the configurations of pieces required to satisfy each rule, in Section 4.3. The associated functions for each class are given in Section 4.4. A detailed example of the use of the move-finding algorithm applied to one particular position is given in Section 4.5. The number of functions associated with each class is 3, including null functions where necessary, and these are denoted by f_1, f_2 and f_3 as in Section 2.

4.2 Specifying the equivalence classes

The rules defining membership of each of the eleven equivalence classes for this endgame and the corresponding class values (held in column 1 of the value table) are summarised in the table below. As explained in Section 2.3, the basic procedure for finding the class of which a given position q is a member is to evaluate each rule in turn until the first true rule is found. This rule gives the class corresponding to position q.

Whenever the definition of a specific rule is discussed it will be assumed that it will only be applied to positions in which all the preceding rules are false. This principle will be used to simplify the definitions themselves wherever possible. Formal definitions of all the rules are given in Section 4.3.

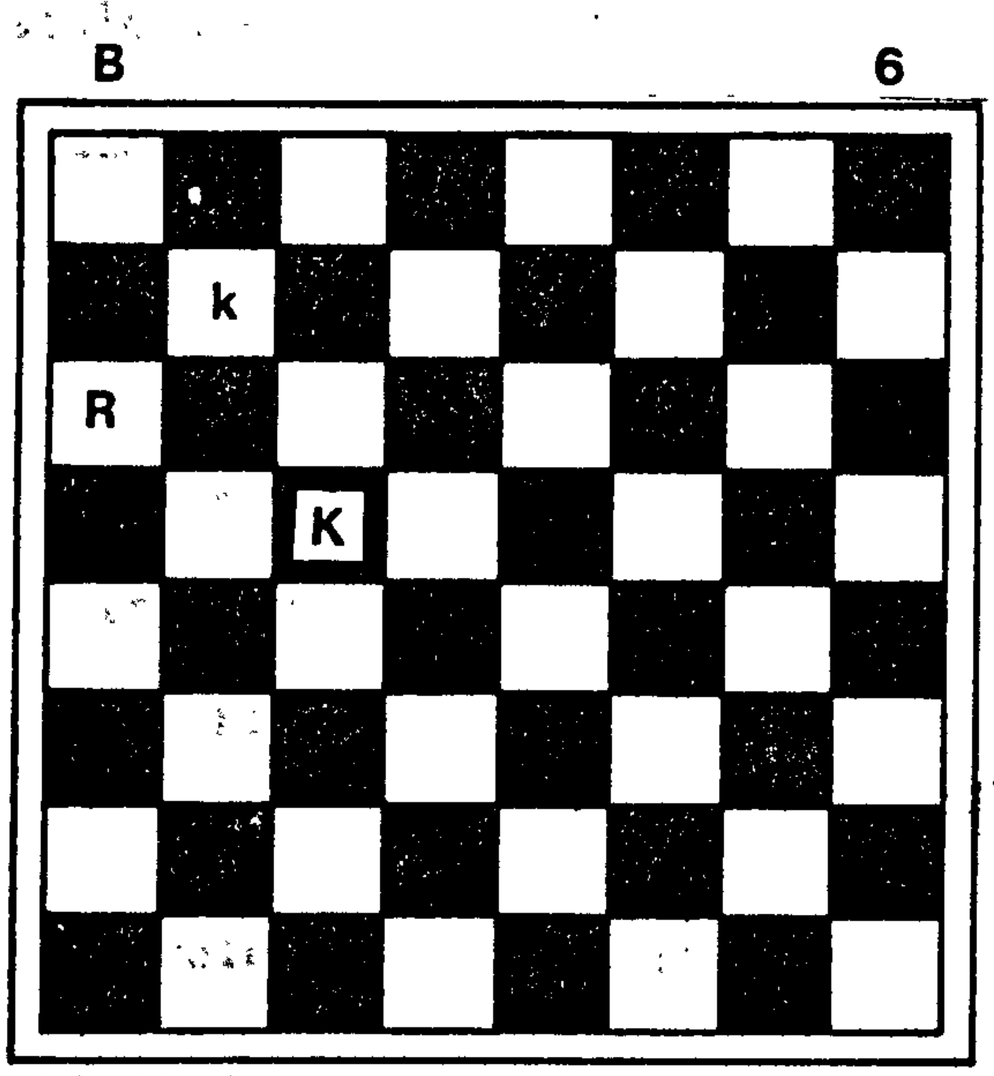
Table 2 Equivalence classes for King and Rook against King (initial algorithm)

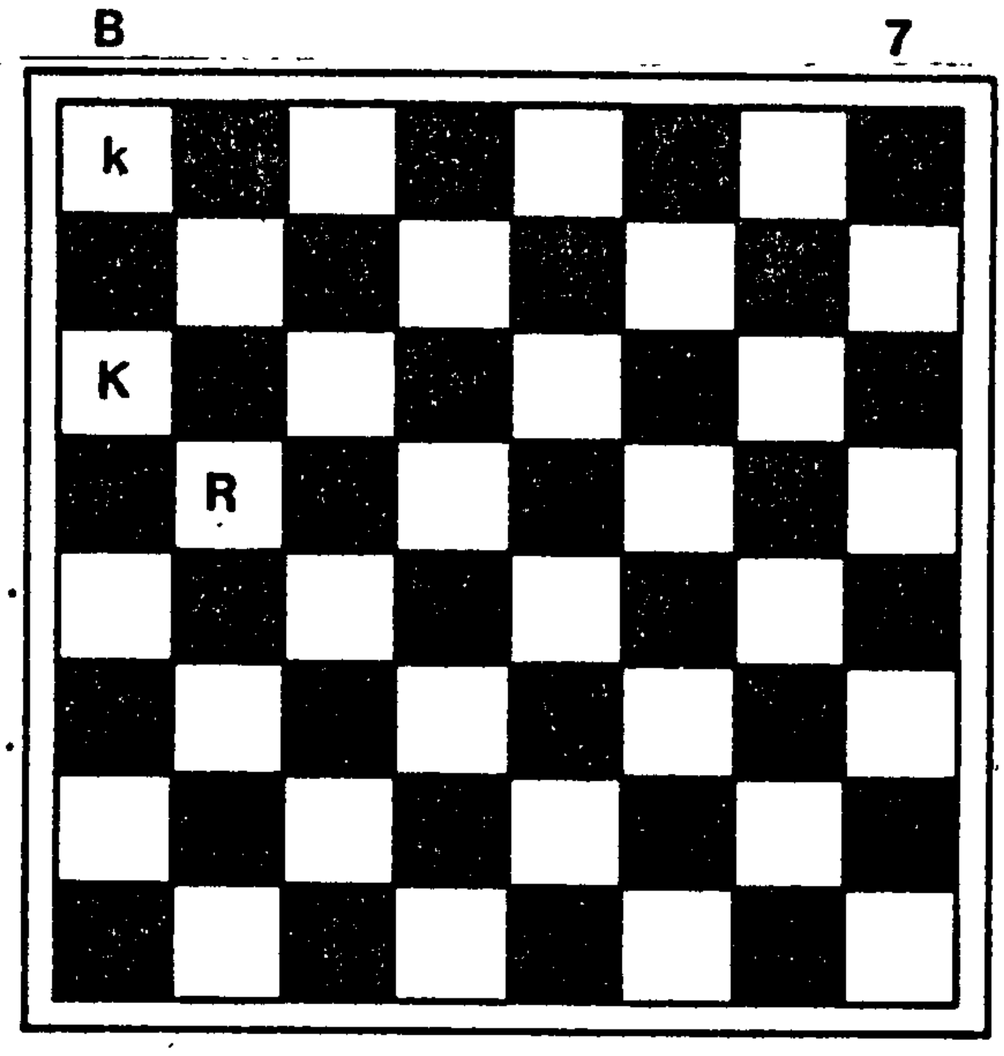
Class	Property of position q (Black to move)	Class Value
1	White's Rook is en prise (i.e. may be immediately captured)	2
2	Black is checkmated	11
3	Black is stalemated	1
4	Black cannot avoid checkmate in one move	10
5	The Kings are in vertical opposition (i.e. they are two squares apart on the same file) and Black is in check along the rank	9
6	The Kings are in vertical opposition with the Rook on the rank between them, one file closer to the centre	8
7	The Kings are two ranks apart with the Rook on the rank between them; the White King is one file closer to the centre than the Black King and the Rook is not precisely one file from the White King	7
8	The Kings are in vertical opposition with the Rook one file closer to the centre	6
9	The Kings are two ranks apart, with the Rook on the rank between them, the Rook controls a "good" quadrant	5
10	The Black King is confined to a quadrant of the board and cannot immediately escape	4
11	(Always true)	3

From the above table it can be seen, inter alia, that positions where Black is checkmated (class 2) are ranked highest of all (all ranking takes place from White's point of view), followed by positions where Black cannot prevent mate in one move (class 4).

Classes 1 and 3 (positions where the Rook is en prise and Black is stalemated, respectively) are ranked lowest. These are the only two classes for which White does not have a forced win against any play by Black. Which of these two classes is ranked the higher is, of course, completely arbitrary. With the order given in the table above, stalemate positions are ranked higher and so White will always prefer to leave his Rook en prise rather than stalemate Black (it can be shown, however, that there will always be other moves preferable to both). Rule 11 has been included to ensure that every position q belongs to some class. Although the "residual" positions which satisfy rule 11 only are intended to be of little significance in this case, class 11 is not the lowest ranked class since it is more favourable than either classes 1 or 3.

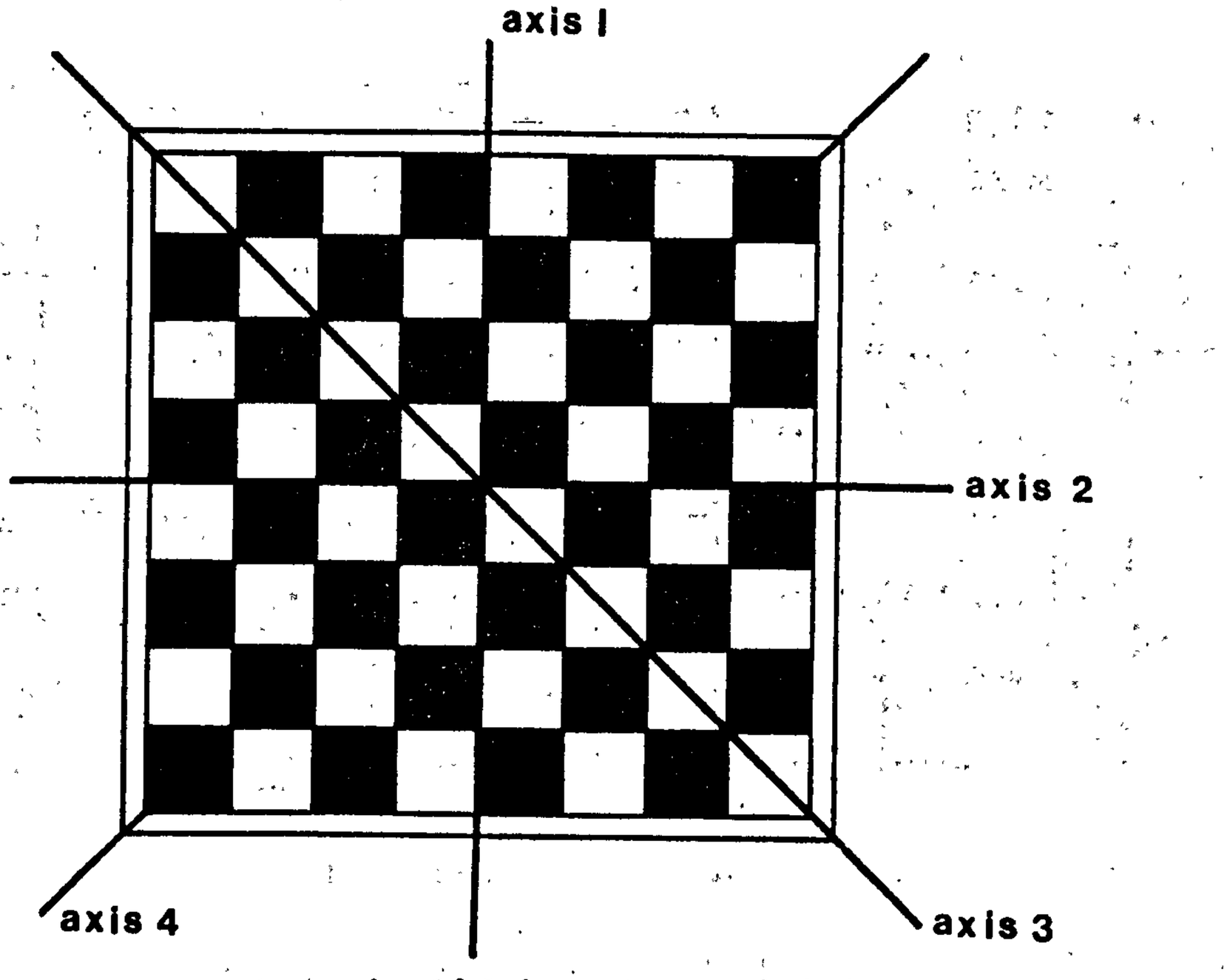
It should be noted that there is nothing in the specification of rule 7 to avoid the possibility of the Rook being en prise (as in Figure 6) and nothing in Rule 8 to avoid the possibility of stalemate (as in Figure 7). This is taken care of automatically by the order in which the rules are tested. Thus Figures 6 and 7 are correctly assigned to classes 1 and 3 respectively.





4.2.1 Symmetry Considerations

In general, every King and Rook against King position has seven symmetrical equivalents, given by any combination of reflections in the four axes shown in the diagram below.

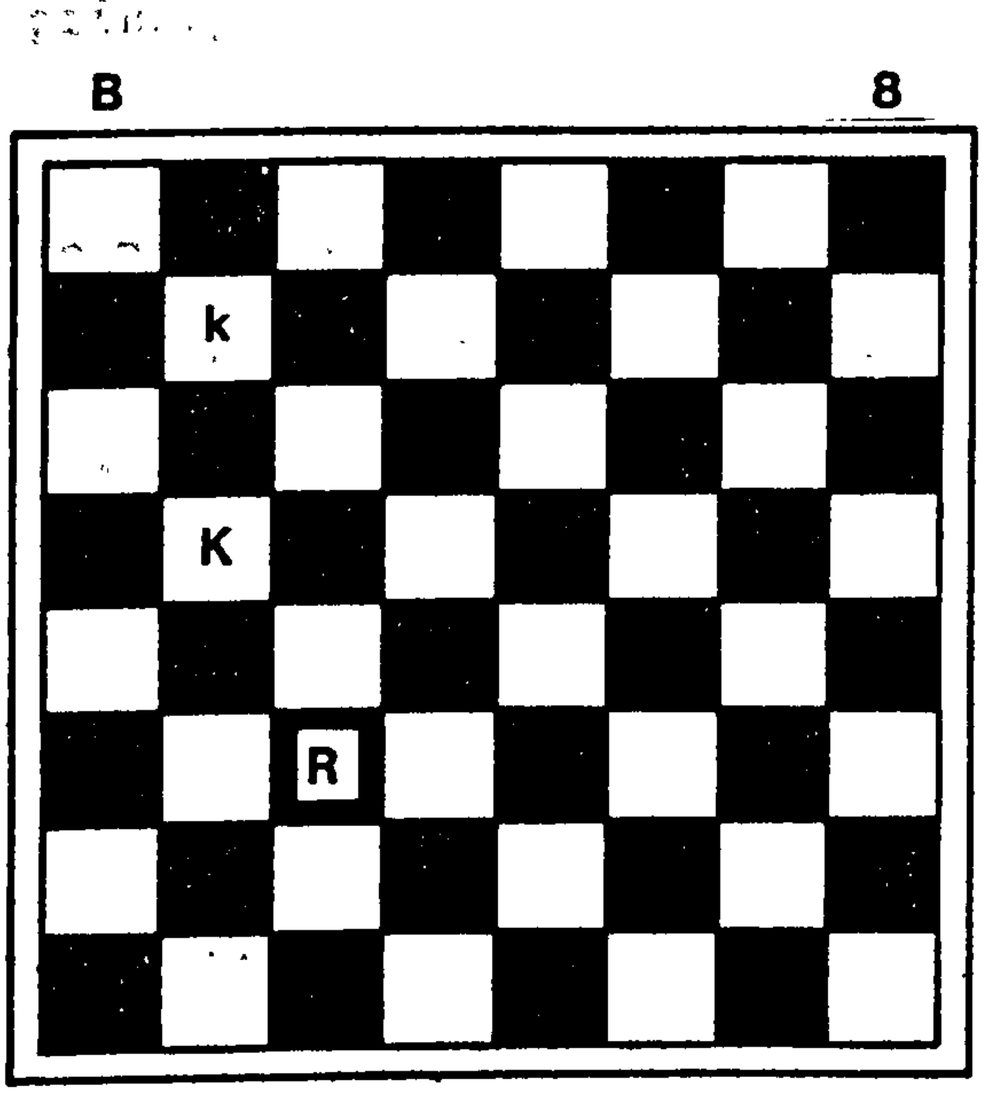


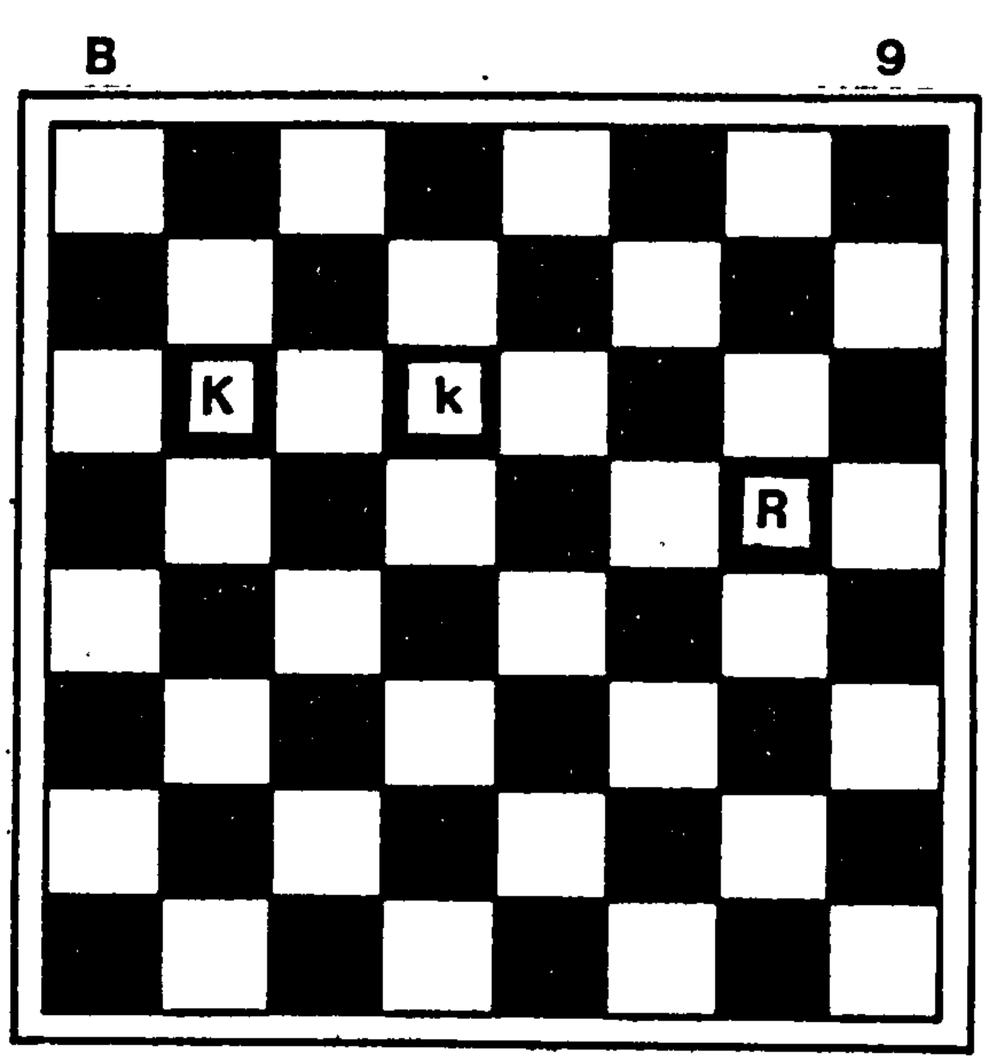
This symmetry, rather than simplifying the definitions of the rules produces additional complications, since, for example, a test for

stalemate positions needs to take into account the possibility of stalemate with the Black King in any one of the four corners of the board.

To allow directly for all such possibilities would make the detailed definitions of the rules unduly lengthy and the following alternative policy has been chosen.

Each rule, generally speaking, is specified and defined for one particular orientation of the board only (such as "Black King on square A8"), although the orientation chosen may vary from one rule to another. Thus, for example, the specification of rule 8 is "the Kings are in vertical opposition with the Rook one file closer to the centre". Figure 8 is an example of a member of class 8.





In order to ensure that other positions, such as Figure 9 (where the Kings are in horizontal opposition, with the Rook one rank closer to the centre) are included, the convention is made that a position q satisfies a given rule if any of the positions equivalent to q by symmetry satisifes that rule. With this convention each class can be defined with a suitable orientation of the board in mind. Thus rule 8

is defined with the Kings in vertical opposition in the left-most four files of the board with the White King below the Black King. Other symmetrically equivalent situations are catered for by the convention given above. With the rule defined in this standard form, its definition is extremely straightforward and is given in Section 4.3.4.

One important consequence of the symmetry convention is that applying any combination of reflections about axes of symmetry to a position has no effect on its class membership. This fact is used in the algorithm to simplify the evaluation of the truth or falsity of many of the rules. The formal definitions in the next section are, in general, given in a suitable standard form. To test whether a rule given in this form is satisfied by a position q it is necessary, in principle, to generate all the positions symmetrically equivalent to q and test whether any one of these positions satisfies the rule. In practice, however, the amount of processing involved can be greatly reduced by a suitable ad hoc combination of transformations and tests. It must be stressed that combining transformations and tests in this way is for purposes of computational efficiency only. In principle, the method previously described is entirely sufficient.

As an example, one way of evaluating rule 8 (table 2) in practice would be as follows.

1. If the Kings are on the same file go on to step 4.
2. If the Kings are on the same rank, reflect the position about axis 3 (or 4) and go on to step 4.
3. Result is false (go on to next rule).
4. If the King's file is in the right-hand half of the board, reflect the position about axis 1.

5. If the White King is above the Black King, reflect the position about axis 2.
6. If the Black King is two ranks above the White King and the Rook is one file to the right of the Kings, then the result is true, otherwise it is false.

The transformations used in this sequence ensure that the standard form for rule 8, as previously described, is achieved. If the Kings are on neither the same file nor the same rank, this will be detected at step 3.

The idea of a standard orientation of the board seems to be one almost invariably (and tacitly) employed in chess textbooks. Thus a phrase such as "restrict the King to the eighth rank" will be used to mean "restrict the King to any edge of the board". The particular orientation chosen may vary from one rule to another. With the particular set of rules used in the algorithm, the number of transformations which need to be applied to any position is generally fairly small, since most of the rules share the same one or two standard orientations.

4.3 Defining the rules

In this section the rules used in the algorithm are formalized and a rationale is given for their choice. None of the definitions to be described makes any use of tree-searching. All the rules are defined solely in terms of the static features of a position. (Once again, it must be emphasized that it is not claimed either that these rules are infallibly correct or that they are the only possible choice which could be made. As will be seen in Section 6, testing the algorithm identified a number of points at which improvements could be made.)

4.3.1 Notation

The notation used for specifying the rules will be that of predicate logic. Although the classes of positions to be described are, it is believed, meaningful to human chess players, neither the vocabulary of the English language nor the notation of predicate logic is ideally suitable for representing them. Predicate logic has been used below since from this form the rules can be readily incoporated in a computer program. The most effective (although informal) form of representation is a combination of diagrams and text and this is the form which is employed in Appendix 1.

The co-ordinates WK1 and WK2 will be used to represent the position of the White King, where WK1 is the number of the file (counting from left to right) and WK2 is the number of the rank (counting from bottom to top) of the square on which the King stands. Thus, if the King is on square H2, then WK1 and WK2 are 8 and 2, respectively.

The co-ordinates BK1 and BK2 will be used for the Black King and WR1 and WR2 for the Rook. (In all cases, the left-hand corner of White's first rank is used as the "origin" of co-ordinates.)

4.3.2 Rules 1, 2 and 3

The inclusion of these rules requires no justification. It should be noted, however, that they are defined entirely by pattern matching, that is, by considering all the possible situations in which "Rook en prise", checkmate or stalemate can occur. This is considerably more efficient than using tree-searching in this case.

Rule 1

To test for the Rook being en prise the function dist is used. This function takes four arguments, the first and second arguments being the file and rank co-ordinates (from 1 to 8) of one square and the third and fourth arguments being the file and rank co-ordinates of another. The value of the function is defined as the larger of the absolute value of the difference in files and the absolute value of the difference in ranks between the two squares.

Tan (1972) calls this value the block distance. It is essentially just the number of King moves needed to move from one square to the other. Using this function, rule 1 is defined by

dist (BK1,BK2,WR1,WR2) = 1 AND dist (WK1,WK2,WR1,WR2) > 1.

In cases where the squares referred to are both occupied by pieces, the notation will generally be abbreviated by using the names of the pieces as arguments, for convenience. Using this convention, rule 1 can be specified by

dist (Black King, Rook) = 1 AND dist (White King, Rook) > 1.

Rule 2

The definition of this rule and those of rules 3-9 all require the Kings to be exactly two ranks apart with the White King on a lower rank than the Black, i.e. BK2 = WK2 + 2.

If the Kings are two ranks apart with the White King above the Black, or if the Kings are two files apart, this orientation can be achieved by a reflection about axis 2, 3 or 4. If the orientation cannot be obtained, rules 2 to 9 must all be false and the next rule which needs to be considered is rule 10. If it can, an additional reflec-

tion about axis 1 is applied, if necessary, to ensure that the Black King is to the left of or on the same file as the White King, i.e. BK1≤WK1, and that if both Kings are on the same file, this lies in the left-hand half of the board.

The only one of the above conditions which will be used explicitly when specifying the definitions of rules 2-9 is BK2 = WK2 + 2; this is the essential property of the positions which needs to be satisfied, the others are matters of orientation only. Once again it should be pointed out that the incorporation of specific transformations of the given position into the algorithm is a matter of computational efficiency only. There is no theoretical requirement for it.

Class 2 consists of positions in which Black is checkmated. It is clear that there is essentially only one kind of checkmate position, with the orientation chosen. The Black King must be in check on the eighth rank with the White King on the sixth rank. The two Kings must be on the same file, except when the Black King is on the Rook file, when a slight variation is possible and the White King can be on the adjacent Knight file instead.

The rule can therefore be written as:

BK2 = WR2 = 8 AND WK2 = 6

AND {WK1 = BK1 OR (BK1 = 1 AND WK1 = 2)}

Rule 3

Stalemate positions (again, with the chosen orientation) can occur only when the Black King is on square A8.

BK1=1 AND BK2=8 AND WK2=6 AND WR1=2

AND {WK1=1 OR (WR2=7 AND WK1≤3)}

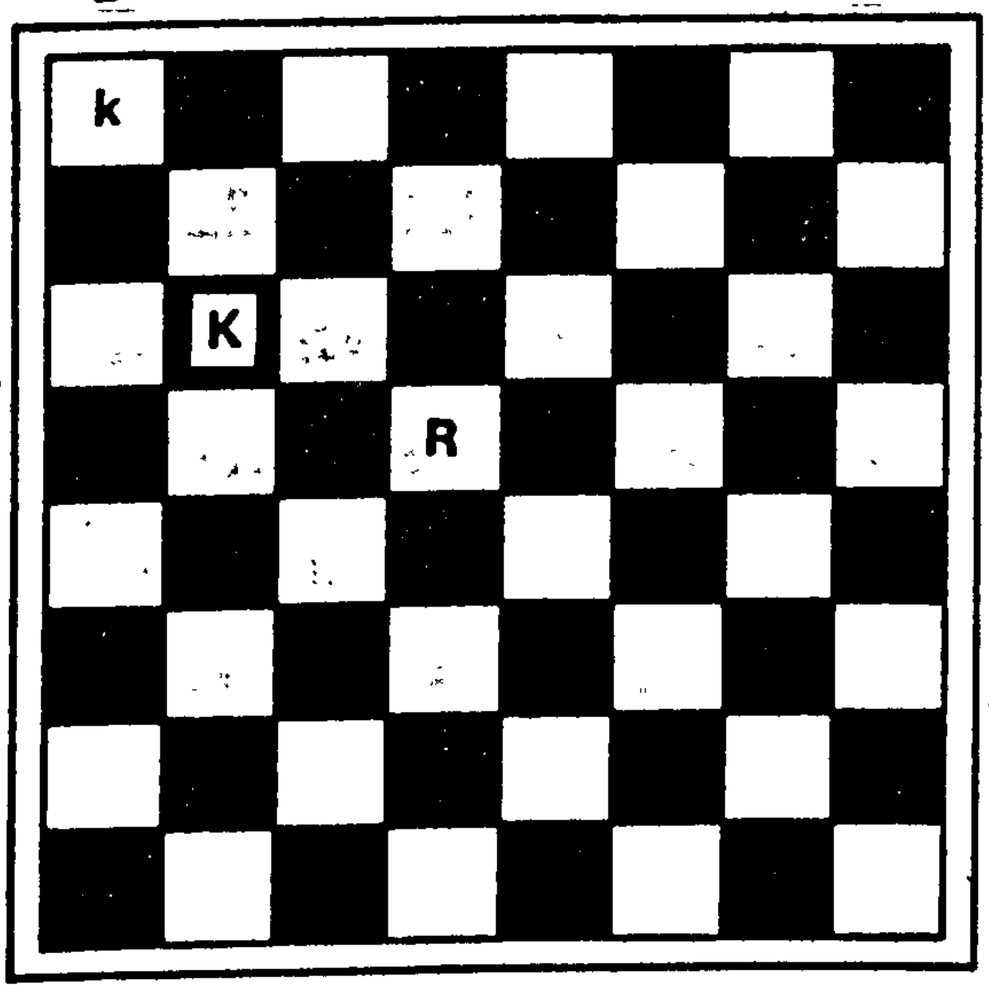
(Note that no tests have been included in either rule 2 or rule 3 to ensure that the Rook is not next to the Black King on the eighth rank and thus en prise. Testing for Rook en prise first makes this unnecessary.)

4.3.3 Rule 4

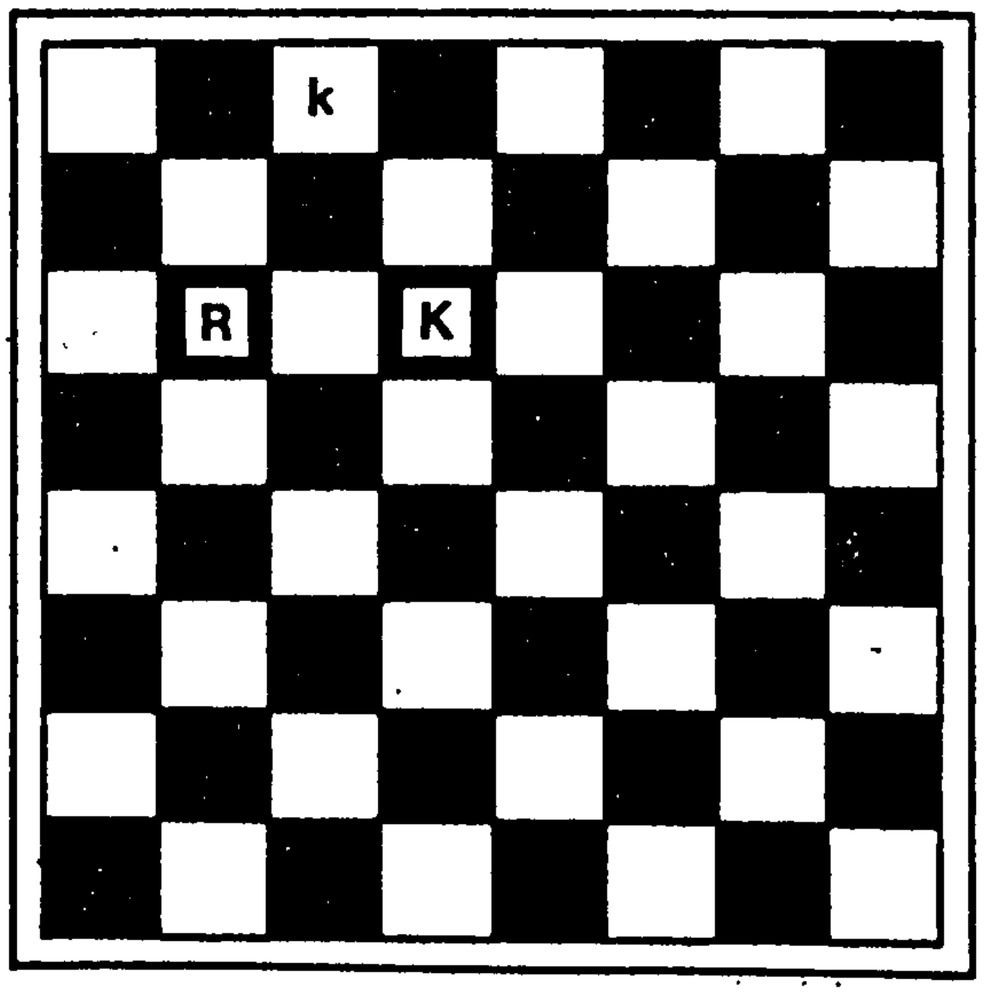
Class 4 contains positions where Black (to move) cannot prevent White from checkmating in one move. In practice, it seems to be helpful to consider these positions as a class in their own right. In many cases White can only win by being specifically aware of the existence of these positions. Thus, for example, Fine (1944) remarks of a White move which leads to such a position "the final manoeuvre, which involves losing a tempo, or move, should be remembered - it is the key to this mate".

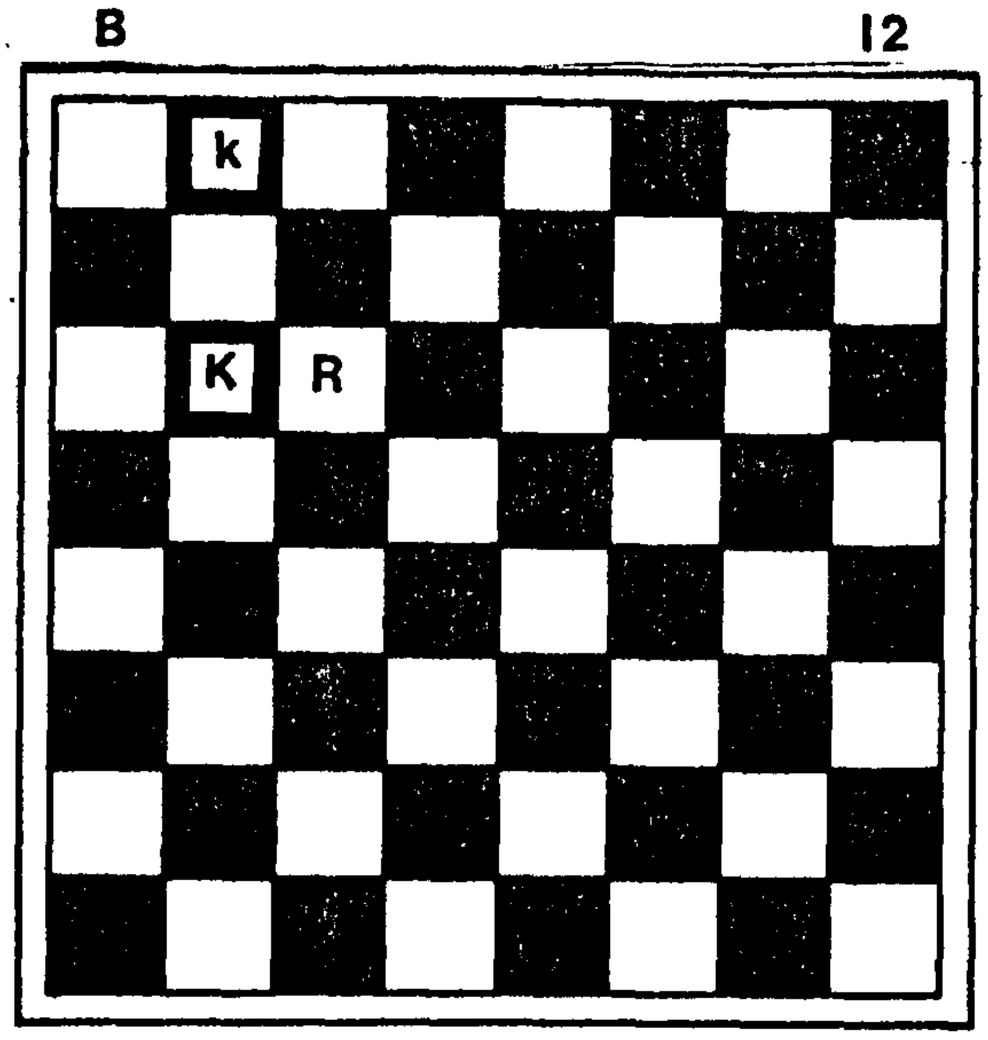
Inspection reveals that there are only three types of position satisfying the rule, exemplified by Figures 10 to 12 below.

B 10



B 11





The rule can be defined by the following fairly complicated expression

```
WK2=6   AND   BK2=8

AND         {(WK1=2   AND   BK1=1   AND   WR1>3)

OR           (BK1=WK1-1   AND   WR1=BK1-1)

OR           (WK1=BK1=2   AND   WR1=3)}
```

(Notice that once again it is not necessary explicitly to rule out the cases where the Rook is <u>en prise</u> or Black is already checkmated. The ordering of the rules takes care of such problems.)

<u>4.3.4 Rules 5 to 10</u>

So far, only the three most fundamental types of position (positions where Black is checkmated or stalemated or where the Rook is <u>en prise</u>) have been defined, together with positions where White can force checkmate in one move.

The remaining rules, however, can only be justified on the basis of an analysis of the checkmating procedure which underlies them. This analysis is based on descriptions given in Chess textbooks but the particular interpretation given to these descriptions can ultimately only be validated by empirical testing, such as that described in Section 6.

Huberman (1968) gives a number of general principles for playing the King and Rook against King endgame, mainly derived from Capablanca (1935) and Fine (1944). Part of Capablanca's advice is reproduced below.

> "The principle is to drive the opposing King to the last line on any side of the board ... [With the Black King confined to the eighth rank, White should advance keeping] his King as much as possible on the same ... file as the opposing King. When the King has been brought to the sixth rank, it is better to place it not on the same file but on the one next to it towards the centre".

The first part of this advice can be rephrased, with a suitable orientation of the board, as "drive the Black King to the eighth rank". The remainder applies to positions where this has already been achieved. Generalising this method to positions where Black has not yet been forced on to the eighth rank gives an overall winning strategy which is summarised in the following informal series of steps.

(i) Confine the Black King to a small number of ranks at the top of the board, using the Rook to set up a barrier along the rank.

(ii) Advance the White King until it is two ranks below the Black King and one file closer to the centre.

(iii) If Black "retreats" a rank, advance the Rook one rank and continue with step (ii).

(iv) If Black remains on the same rank, but moves towards the edge of the board, move the White King one square towards the edge also, still keeping two ranks apart and one file closer to the centre. This will eventually force Black to choose either (iii) or (v).

(v) If Black takes the opposition (i.e. moves his King to the same file as the White King and two squares apart), force him to move back a rank by a check with the Rook and continue with step (ii).

Rules 5 to 10 represent a formalisation of these steps.

The class values for Classes 5 to 10 are chosen to reflect the order of priority of White's goals. White's highest priority in the above series of steps is to force Black's King back by a check where possible (step (v)).

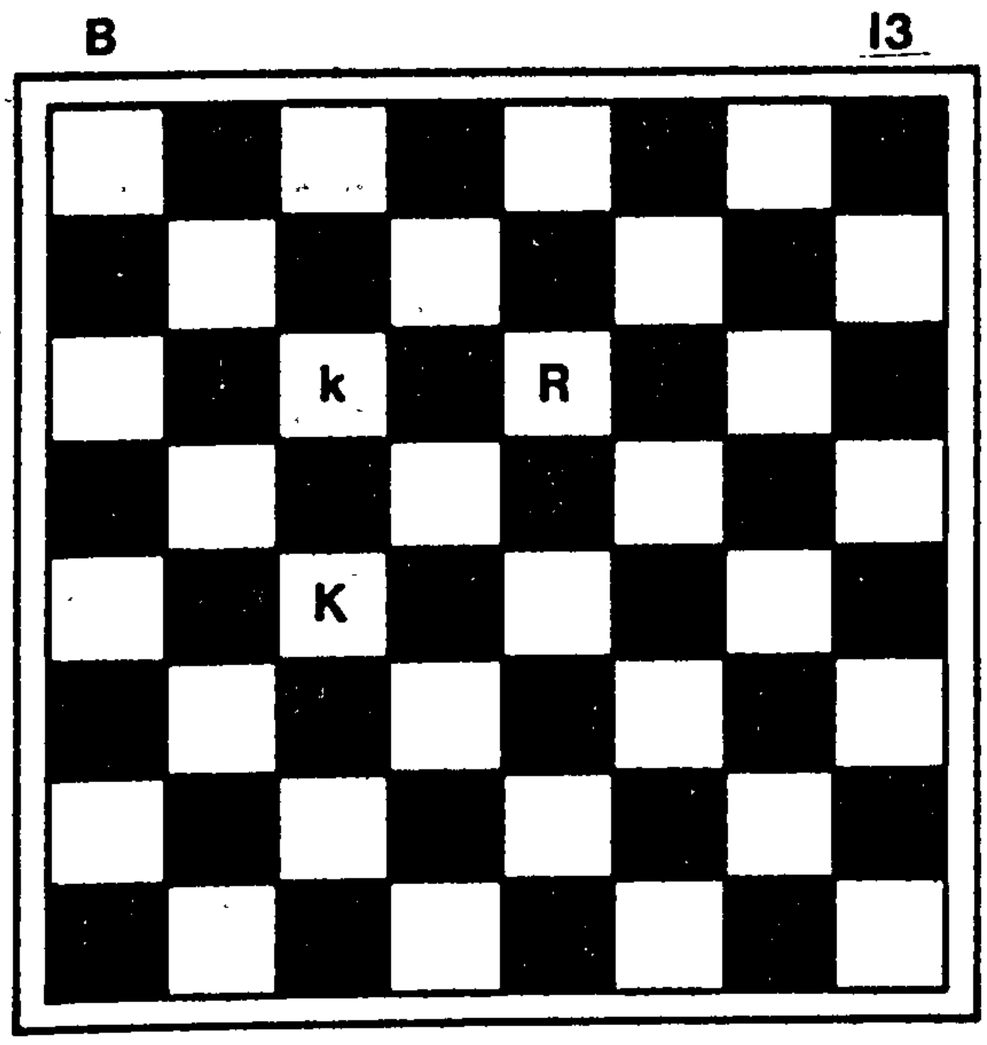
Step (i) on its own is the lowest priority.

Rules 5,7 and 10 are directly related to the above steps. Rules 6, 8 and 9 have been included to cater for some other important situations which can arise when implementing White's general strategy.

The most significant of these seems to be the importance for White of maintaining a position with the Kings two ranks apart, and the Rook on the rank between them (although this is not his highest priority). This point is not usually explicitly mentioned in text-books (see for example Fine (1941 and 1944), Golombek (1954) and Capablanca (1935)).

Rule 5

Rule 5 corresponds to step (v) of the above description. Class 5 consists of positions where the Kings are in vertical opposition and Black is in check along the rank (see Figure 13).



When the Black King is on the Rook file, the White King can be on the adjacent Knight file instead, as in Figure 14.

The complete rule can thus be written as

(BK2=WK2+2) AND (WR2=BK2)

AND {WK1=BK1 OR (BK1=1 AND WK1=2)}

This rule is, in fact, identical to rule 2, except that the Black King and the Rook are not on the eighth rank.

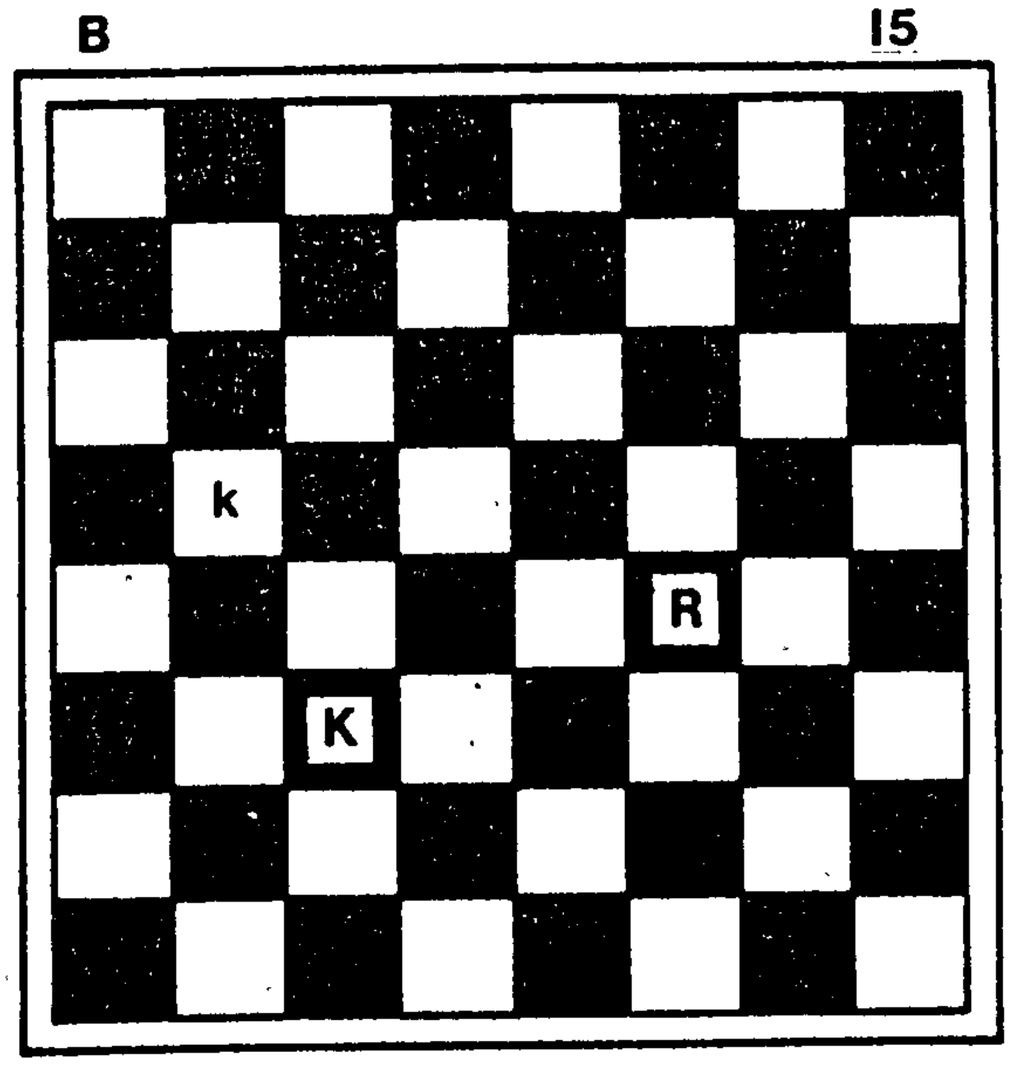
Rule 7

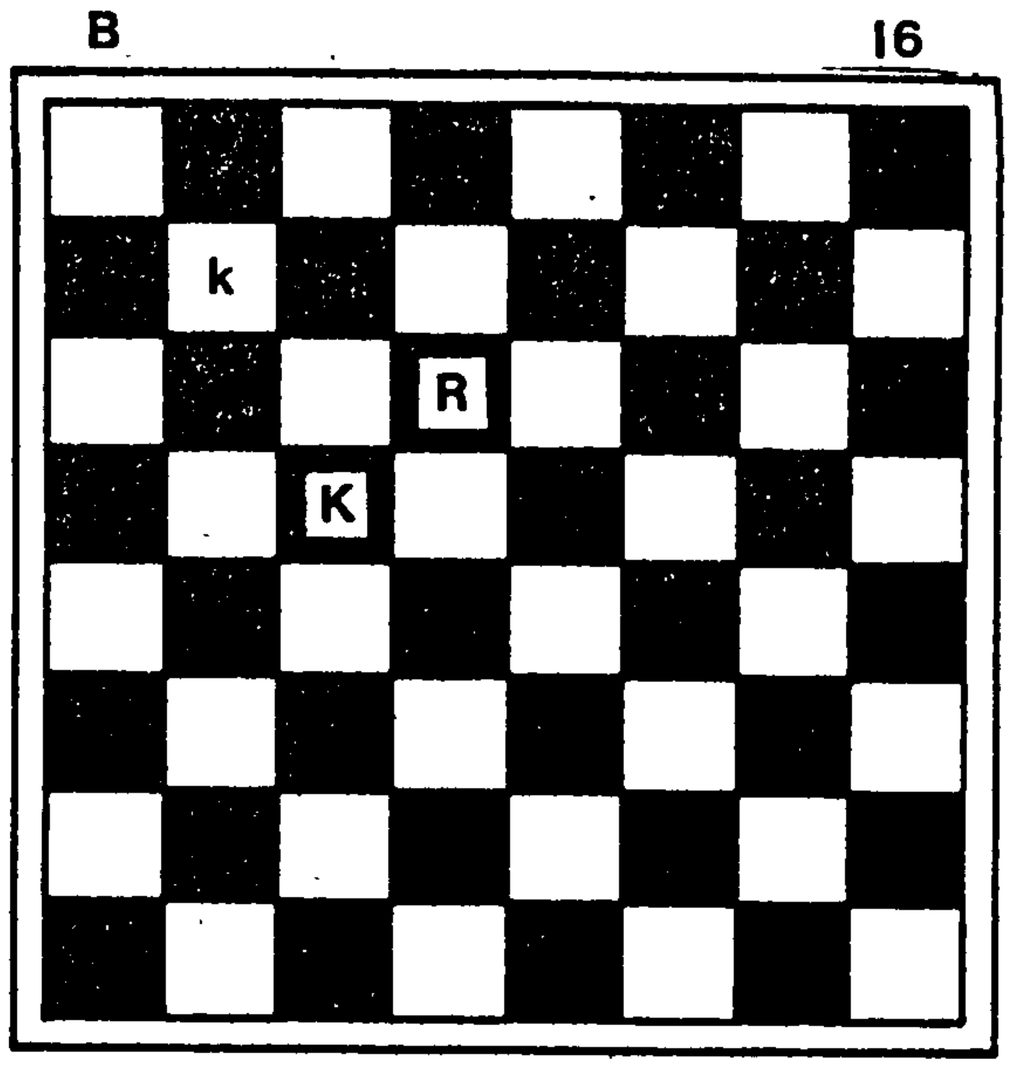
Ignoring rule 6 at present, rule 7 corresponds to the "chasing" process in steps (ii) and (iv) of the description of White's strategy.

The Kings must be two ranks apart with the Rook on the rank between them. For simplicity this condition (BK2=WK2+2 AND WR2=WK2+1) will be denoted by the predicate function chase. (This function will also be used by some of the rules which follow.) Apart from the function chase being true, the White King must be one file away from the Black King and should be closer to the centre, since the intention is to chase Black towards the nearest (vertical) edge of the board. The case where the White King is on one centre file and the Black King is on the other will be included, since Black will then be "forced" to move towards an edge. This gives the further condition

BK1≤4 AND WK1=BK1+1.

Figure 15 is an example of a position in class 7. Black must now give ground, for example by K-A5. If he plays to C5 instead, he will be forced back by a check (R-F5).





The definition of rule 7 is still incomplete, however. It is a fundamental requirement that if Black should play from the specified position on to the same file as the White King (still two ranks apart), White should be able to force him back with a check. For this reason it is necessary to exclude positions where the Rook is one file to the left or one file to the right of the White King. Thus, for example, Figure 16 must be excluded since, if Black now plays K-C7, White cannot force him back by R-D7 ch, since then the Rook is <u>en prise</u>. Putting all the above considerations together, rule 7 can now be defined by

<u>chase</u> AND BK1≤4 AND WK1=BK1+1

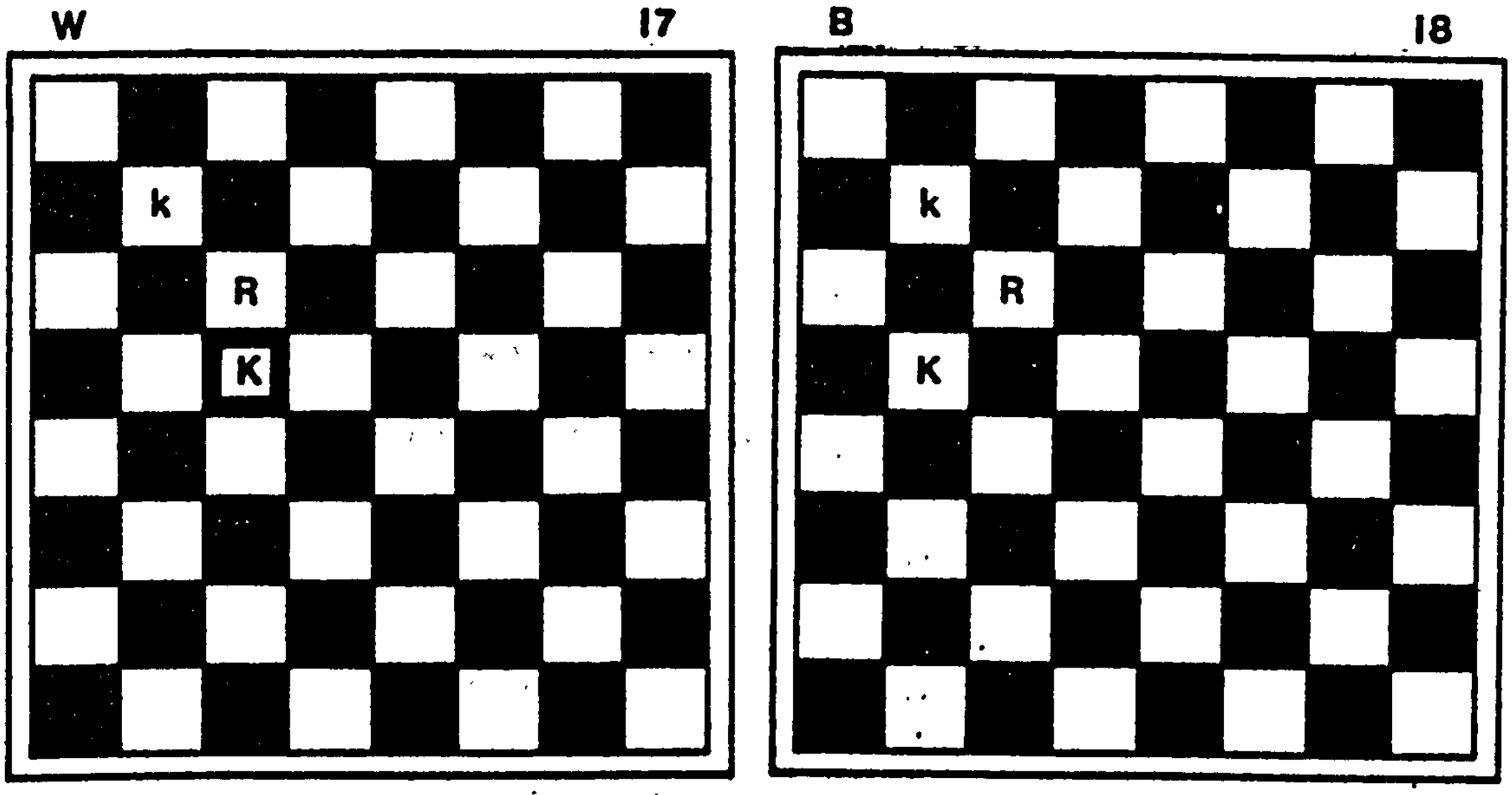
AND <u>abs</u> (WK1-WR1)≠1

where <u>abs</u> is the absolute value (or modulus) function.

(Note that on some occasions the Kings may be two files as well as two ranks apart with the Rook on the <u>file</u> between them. In this case, a reflection about axis 3 will make <u>chase</u> true and maintain the desired orientation as before.)

Rule 6

Class 6 has been included (ranked higher than class 7) to allow for positions such as Figure 17.



In this position White's best move is probably K-B5 not R-H6, say, which would give a position in class 7. Class 6 contains positions, such as Figure 18, where <u>chase</u> is true and the two Kings are on the same file with the Rook one file closer to the centre (including the case where the Kings are on one centre file and the Rook is on the other).

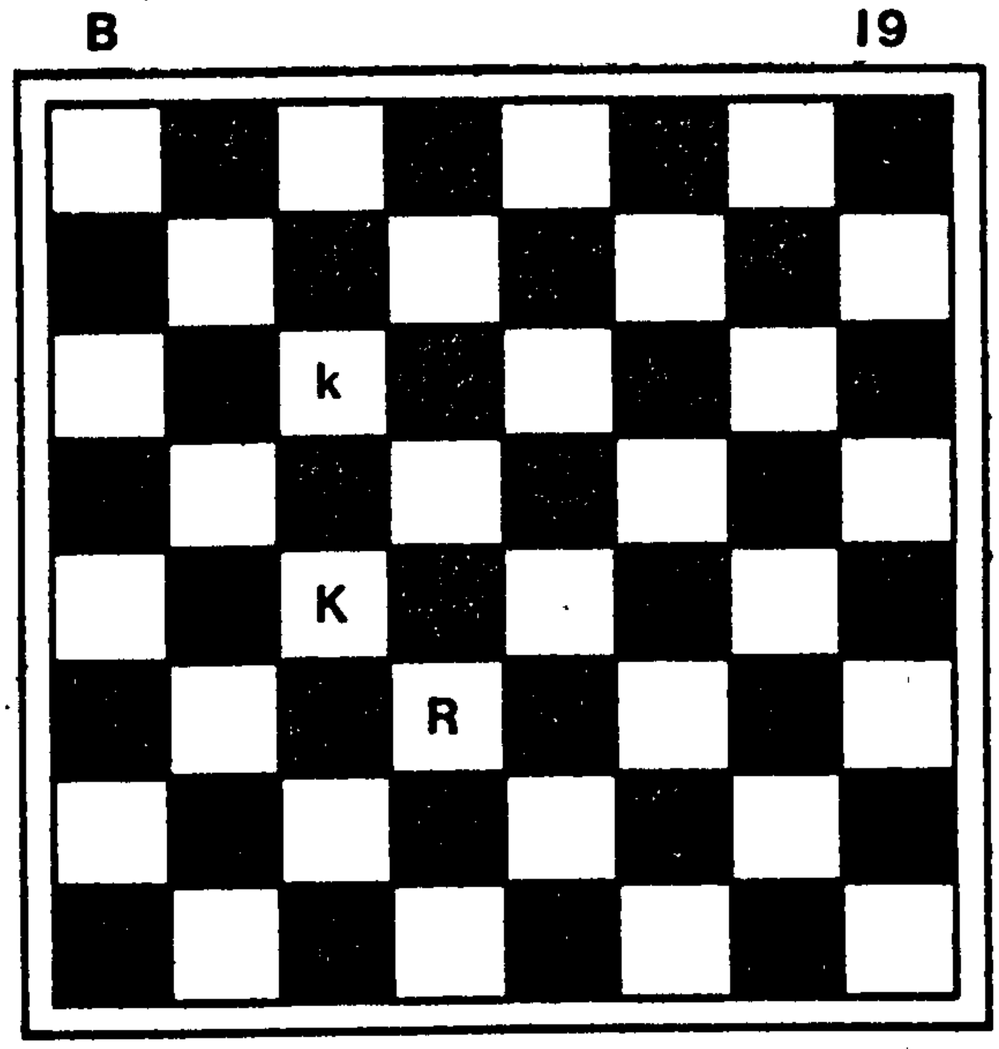
Rule 6 can be defined as follows

<u>chase</u> AND (WK1=BK1) AND (WR1=WK1+1)

Rule 8

Class 8 consists of positions similar to those in class 6, except that the Rook is not on the rank between the two Kings. (If it were, the positions would be members of class 6 instead.)

Figure 19 is an example of a member of class 8.



Rule 8 is defined as follows:

(BK2=WK2+2) AND (WK1=BK1) AND (WR1=WK1+1).

Rule 9

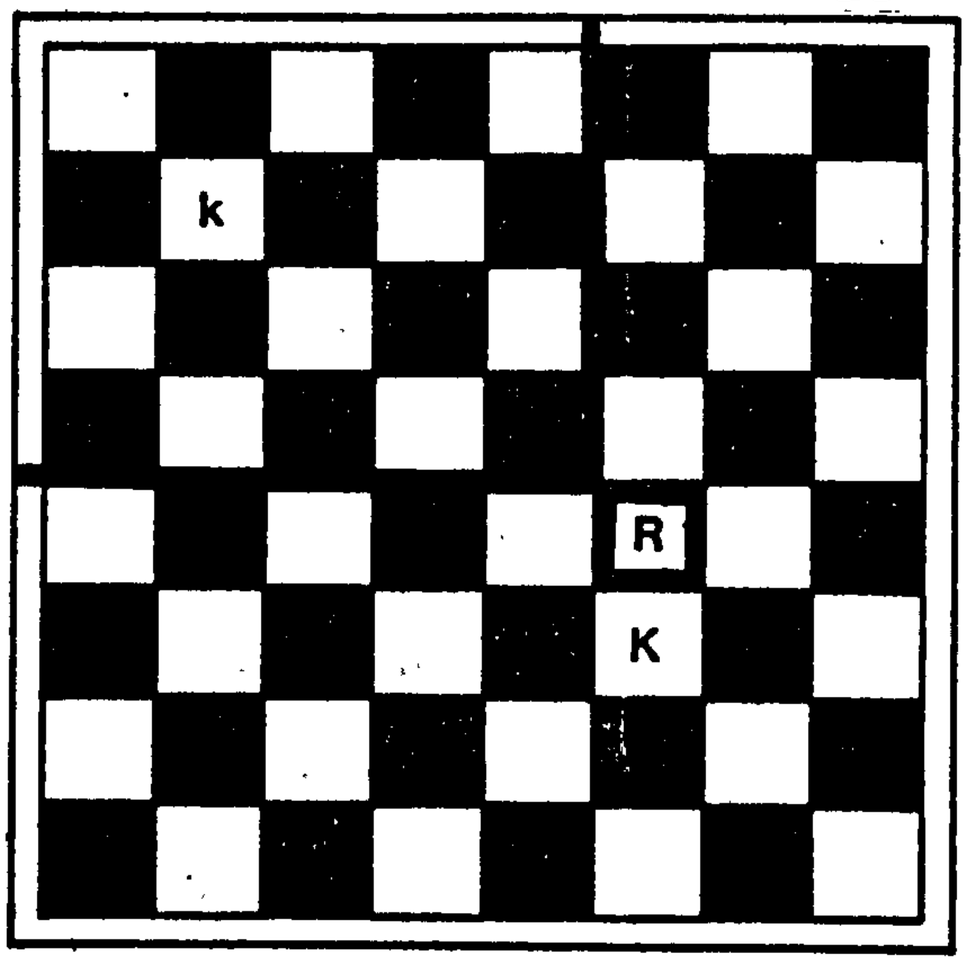
Class 9 contains all the positions not in any of the previous classes where chase is true and the Rook controls a "good" quadrant. The notion of a "good" quadrant used by the program is based on ideas developed by Huberman (1968), with some modifications. Huberman introduces the concept of the Rook controlling a quadrant of the board in the following words:

"as long as the Black King is not in check it is held in some area of the board by the Rook [called a quadrant]. ... If the White King is not on the boundary of the area, the Black King can escape only by attacking the Rook. If the White King is outside of the area ..., it is able to protect the Rook from such an attack if it is close enough. It can't be blocked from protecting the Rook by the Black King".

Thus, in Figure 20, the Rook's quadrant comprises the 20 squares which lie in the area indicated (ranks A to E, files 5 to 8, inclusive).

Following Huberman, a function squad is defined, the value of which is the number of squares inside the quadrant (thus squad is 20 for Figure 20). This function will be used in Section 4.4.

20



For the purposes of the move-finding algorithm the Black King is restricted to being entirely inside the Rook's quadrant, that is, it must not be on the same rank or file as the Rook (Huberman's condition that Black should not be in check is somewhat weaker than this). The White King must be completely outside the quadrant.

The quadrant will be called "good" provided that the Black King is not closer to the Rook than the White King (since otherwise it could attack the Rook and force it to move away). This condition will be denoted by the predicate goodquad (again using Huberman's terminology, but with slight differences of definition).

Rule 9 can now be defined by

chase AND goodquad.

This rule ensures that Black is restricted to a fairly small area of the board and that he cannot attack the Rook and force it away. (The associated functions described in Section 4.4 are used to ensure that the area is chosen to be as small as possible.)

All that remains is to define the predicate function goodquad. Since the conditions defining chase must always be satisfied for this rule to be true, the White King can automatically be taken to be outside the Rook's quadrant and the definition of goodquad for this present purpose can thus be simplified accordingly.

To ensure that the Black King is actually inside a quadrant (i.e. not in check along the file) a standard orientation of the position with BK1≤WR1 (i.e. with the Black King on the left of or on the same file as the Rook) is taken, applying a reflection about axis 1 if necessary.

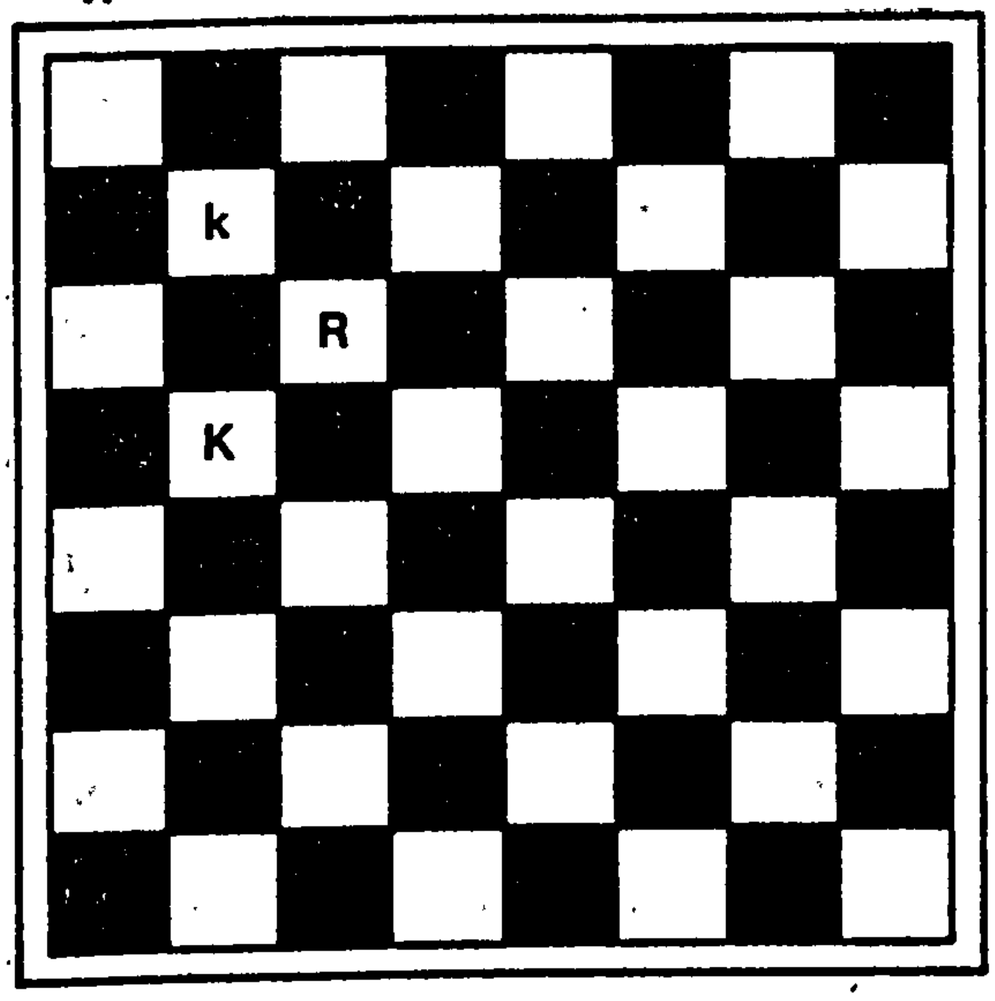
(After this reflection the White King need no longer be to the right of or on the same file as the Black, but this fact will not be significant.) The King will then be inside the quadrant if BK1<WR1.

The definition of goodquad can now be written as

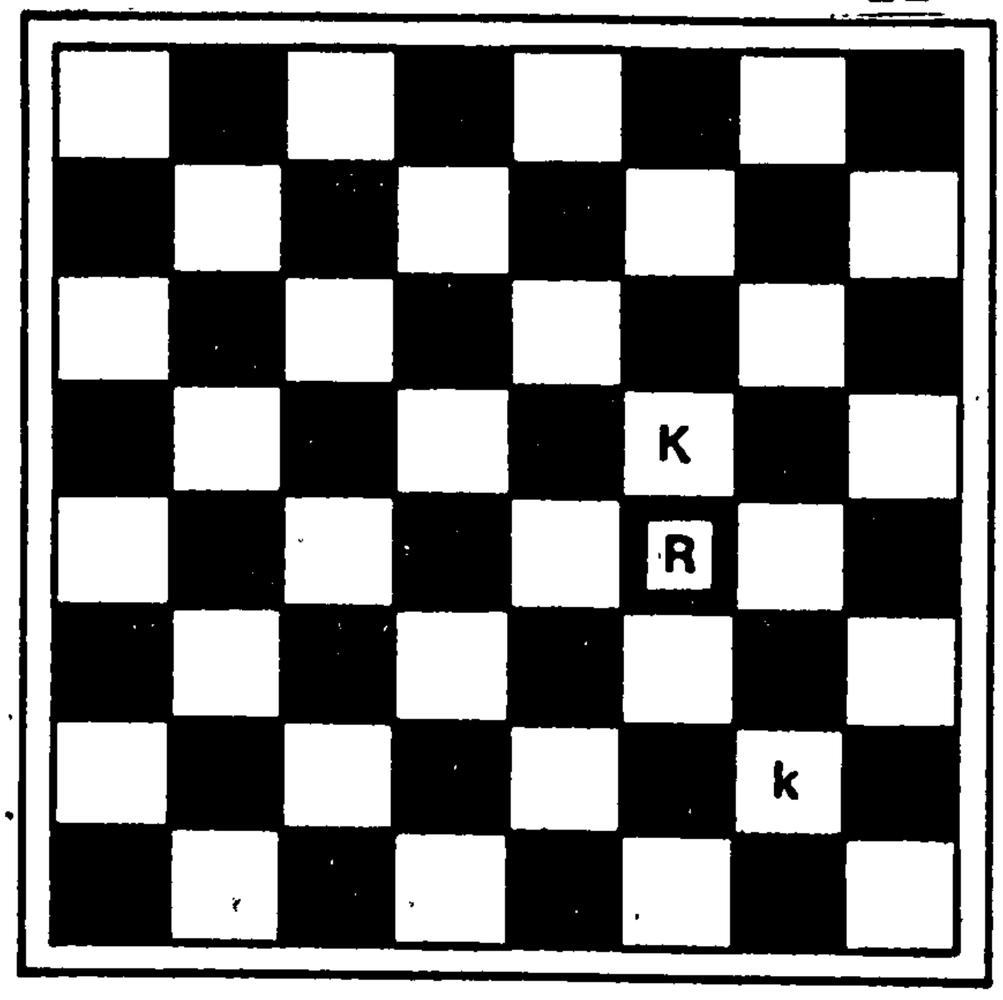
BK1<WR1 AND dist (Black King, Rook) ≥ dist (White King, Rook)

where dist is the "block distance" function used for rule 1.

W 21



W 22



Considering classes 5 to 9 as a whole, class 5 is given the highest class value since it corresponds to the most important step (v) of White's strategy.

The example in Figures 17-18 demonstrates the need for class 6 to be ranked above class 7, and Figures 21 and 22 are given here to illustrate the order of ranking chosen between classes 7,8 and 9 as well as the significance of the inclusion of the two latter classes in the algorithm.

In Figure 21, it would seem that White should play K-C5 giving a class 7 position, in preference to R-C5 (class 8). In Figure 22, White should probably choose K-G4 (class 8), not K-E4, E5 or E6 (class 9). The overall order of ranking between classes 5-9 is 5(highest), 6,7,8,9 (lowest).

Rule 10

Class 10 contains all the significant positions not previously mentioned. It corresponds to steps (i) and (iii) of the strategy given above, i.e. "confine the Black King to a small number of ranks at the top of the board, using the Rook to set up a barrier along the rank". In fact, however, the rule and its associated functions serve to cause Black to be confined to a limited number of ranks or files, whichever is the smaller in any particular situation.

The rule itself merely ensures that Black is confined to a quadrant, the associated functions are used to ensure that the number of ranks to which he is restricted (or the number of files, whichever is the smaller) is as small as possible.

The rule permits the White King to be anywhere on the board with the sole exception that it may not be on the same rank or file as the Rook on the rare occasions when this would enable the Black King to move on to the same line, using the White King as a shield against the action of the Rook, and thus leave the quadrant.

Taking a standard orientation with BK1≤WR1 and BK2≥WR2 (i.e. the Black King not to the right of and not below the Rook), applying reflections about axes 1 and 2 if necessary, the rule can be defined as follows:

WR1>BK1 AND WR2<BK2

AND NOT (WK2=WR2 AND BK2=WR2+1 AND BK1<WK1<WR1)

AND NOT (WK1=WR1 AND BK1=WR1-1 AND BK2>WK2>WR2)

(Note that the condition BK2=WK2+2 and the orientation described under rule 2 do not apply to this rule.)

4.3.5 Rule 11

Class 11 contains all the "residual" positions which do not belong to any of the other classes. In this case, it is expected that the successor position selected as "best" by the algorithm will seldom or never belong to this class, since the definition of rule 10 is so broad that it is likely that White will at worst always have some move at his disposal which gives a successor position in class 10. The definition of rule 11 is simply the logical value true.

4.4 Defining the associated functions

As described in Section 2.4, the ranking between positions in the same equivalence class is determined by means of the associated functions for each class. For this endgame, a total of five functions are defined, each having an index number in the range 1 to 5 inclusive. The five functions are defined below. They are all relatively simple geometric properties of a position.

For convenience, the definitions of functions 1 and 2 have been simplified by assuming that the Black King is confined to a quadrant above the rank of the Rook. This condition is implied by the definitions of the classes (7,9 and 10) with which the functions are associated. Once again this is purely a matter of computational convenience.

Function 1 is used to ensure that the number of ranks or files (whichever is the smaller) to which the Black King is restricted by the Rook is as small as possible, for class 10. It is defined by

8 - min ((WR1-1), (8-WR2))

Function 2 is used to ensure that the Black King is confined to as small a quadrant as possible, for classes 7 and 9. It is defined by

50 - squad

where squad is the number of squares in the quadrant occupied by the Black King.

Squad can be calculated by

(8-WR1) × (8-WR2) where WR1<BK1

and (WR1-1) × (8-WR2) where WR1>BK1

Function 3 is used to make the file difference between the Kings as small as possible, for class 9. It is defined by

8 - abs (WK1-BK1)

Function 4 is used to make the block distance between the Kings, i.e. the larger of the file difference and the rank difference, as small as possible, for class 10. It is defined by

$$8 - \underline{\max}\ \{\underline{\text{abs}}(WK1-BK1),\ \underline{\text{abs}}(WK2-BK2)\}$$

Function 5 is used to make the smaller of the rank difference and the file difference between the Kings as small as possible, for class 10. It is defined by

$$8 - \underline{\min}\ \{\underline{\text{abs}}(WK1-BK1),\ \underline{\text{abs}}(WK2-BK2)\}$$

The functions have been defined so that their values are all positive or zero integers. The value of function 2 is less than 100, the values of the others are less than 10. A suitable choice for the constant K, introduced in Section 2.6, is therefore 100.

4.4.1 The functions associated with each class

For each equivalence class, the index numbers of the associated functions are held in the corresponding row of the value table; functions f_1, f_2 and f_3 being held in columns 2, 3 and 4, respectively.

The complete value table for this endgame is shown below.

Table 3 Value table for King and Rook against King (initial algorithm)

Row/Column	1	2	3	4
Class:	Class value	f_1	f_2	f_3
1	2	0	0	0
2	11	0	0	0
3	1	0	0	0
4	10	0	0	0
5	9	0	0	0
6	8	0	0	0
7	7	2	0	0
8	6	0	0	0
9	5	2	3	0
10	4	1	4	5
11	3	0	0	0

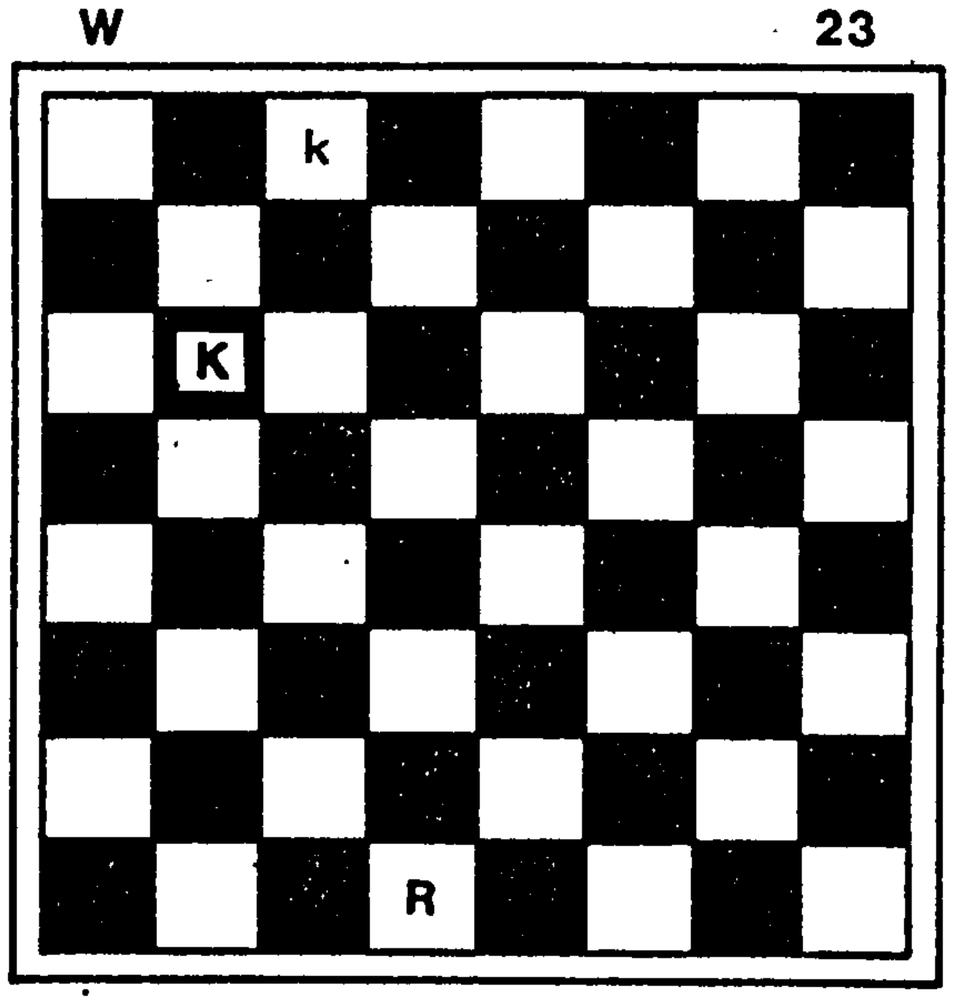
(A zero entry denotes a "null function", i.e. a function with value zero for any position.)

From the above table it can be seen that only three of the equivalence classes (classes 7, 9 and 10) have associated functions, although for a more complicated endgame, it is possible that many or all of the classes will have associated functions.

Functions f_1, f_2 and f_3 are principally used to discriminate between two successor positions belonging to the same class. Since it can be proved that there are no positions from which White has a choice of two different checkmating moves, no discrimination is needed for class 2. Similar considerations can be shown to apply for classes 5 and 6. Although in some positions White can choose more than one way

to stalemate Black or leave his Rook en prise, there are always better moves available. Thus, a class 3 or class 1 position will never be chosen as best so there is no particular advantage in discriminating between positions in those classes. Similarly, there is little point in discriminating between class 4 positions (where Black cannot avoid mate in one). In Figure 23, for example, White can achieve a position in class 4 by any one of the five moves R-D6, R-D5, R-D4, R-D3 or R-D2, since after any of these Black must play K-B8 whereupon White wins by R-D8 mate. All of these five initial moves are equally good, by any non-arbitrary criterion, however it would be possible to ensure that White played 1.R-D6 not 1.R-D5 (or D4, D3 or D2) simply by defining a new function (function 6) with value WR2 (that is, the number of the rank on which the Rook stands), and entering a 6 in column 2 of the fourth row of the value table.

W 23



A similar method could be used to discriminate between positions in classes 8 and 11, but no practical cases have so far been found in which such discrimination would be worthwhile.

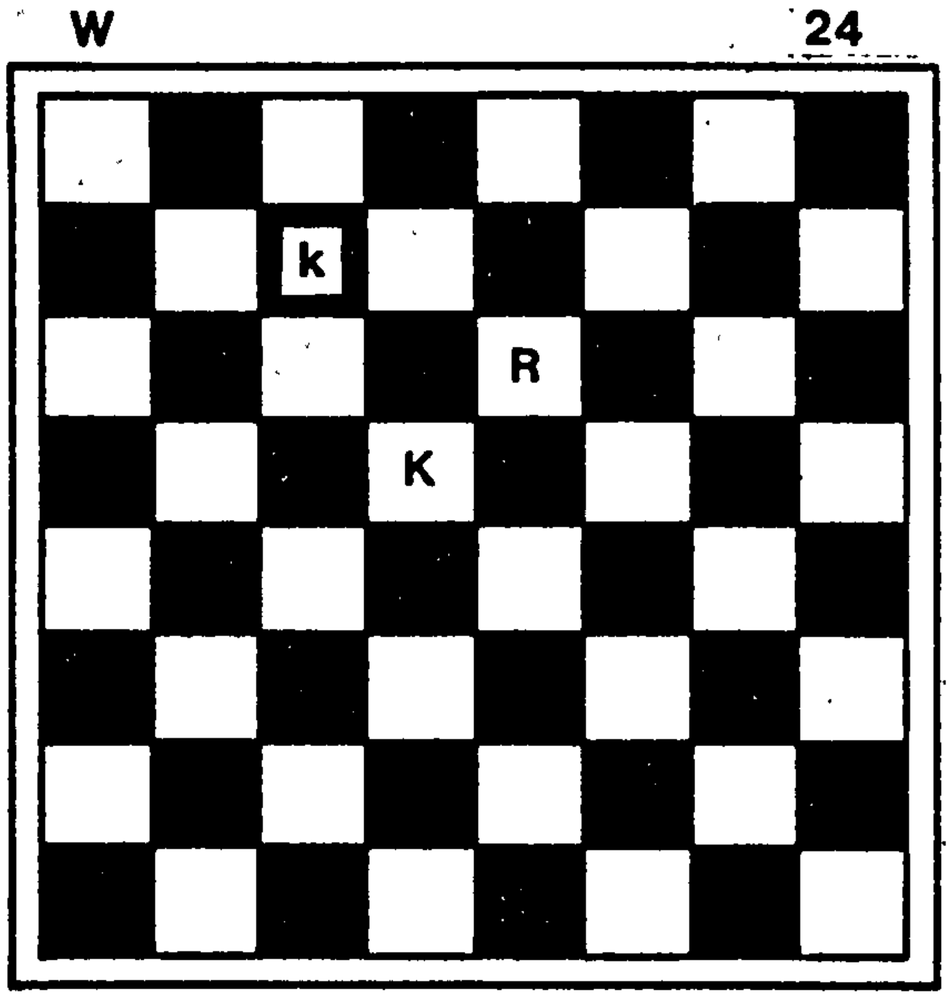
The functions used to discriminate between positions in classes 7, 9 and 10 will be described in detail in the following sections.

4.4.2 Class 7

Discrimination between positions in class 7 is made on the basis of function 2 only, that is the position is chosen which gives the smallest value of squad (the number of squares in the quadrant to which the King is confined - the definition of class 7 ensures that the King must be confined to a quadrant, not necessarily a "good" one).

Using this function, White will play 1.R-D6 in Figure 24 (giving a quadrant of 6 squares), in preference to 1.R-F6 (giving a quadrant of 10 squares).

W 24

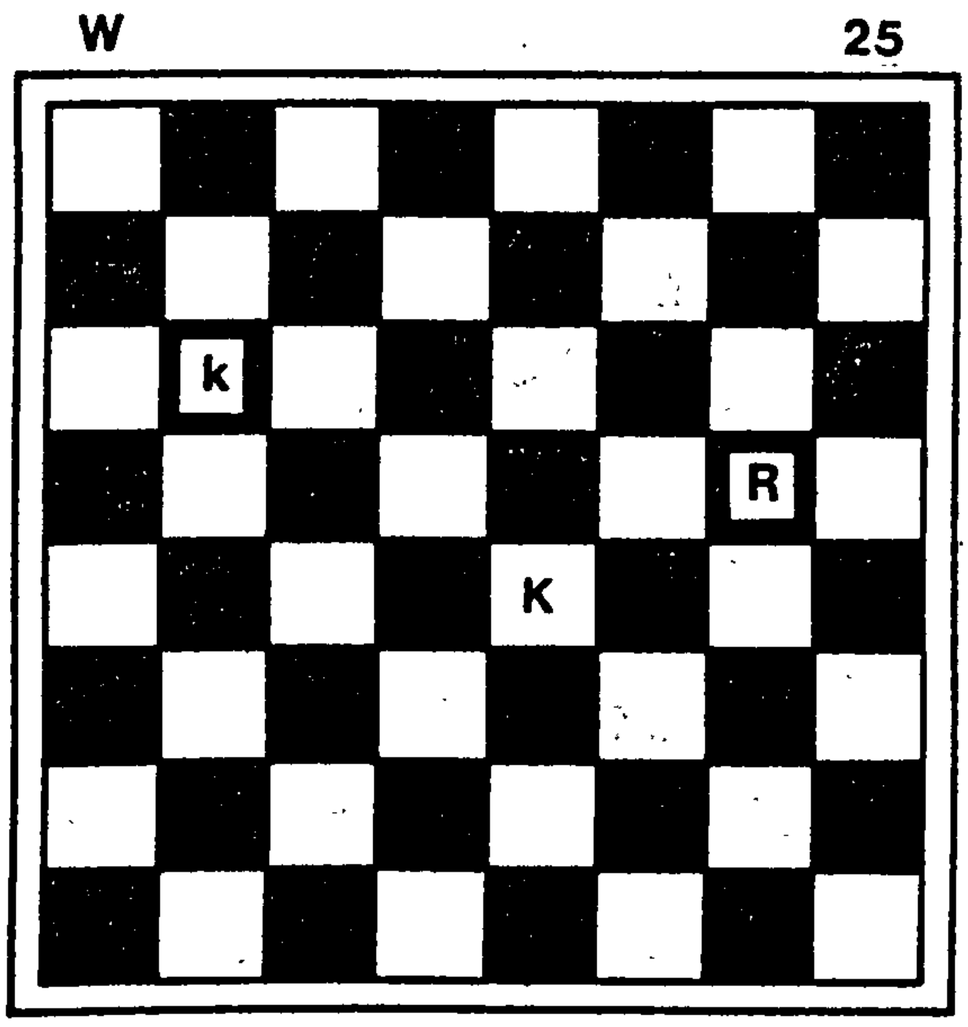


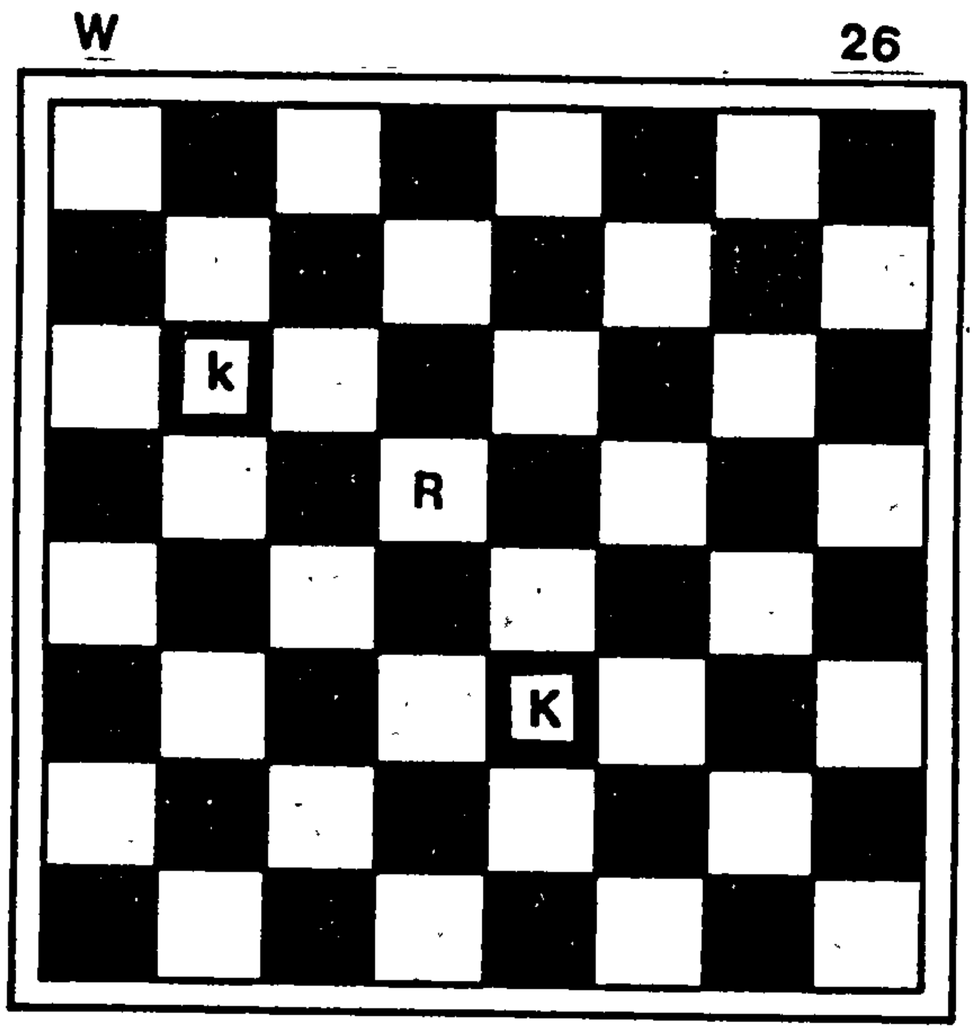
4.4.3 Class 9

Discrimination between positions in class 9 is made on the basis, firstly, of choosing the position with the smallest value of squad (function 2) and, if this is not sufficient to discriminate between the positions, secondly of choosing the position with the smallest value of the file difference between the two Kings (function 3).

Thus in Figure 25, White plays 1.R-D5, restricting Black to a quadrant of 9 squares. Playing 1.R-E5, F5 or H5 would give a larger value of squad.

In Figure 26, White chooses 1.K-D4 not 1.K-E4. Either move produces a position in class 9, with the same value of squad, i.e. 9. For the move 1.K-D4, however, the file distance between the Kings is 2 compared with 3 for 1.K-E4. Thus 1.K-D4 is chosen.





4.4.4 Class 10

Discrimination between positions in Class 10 is made by choosing the position which has the smallest value of the following:

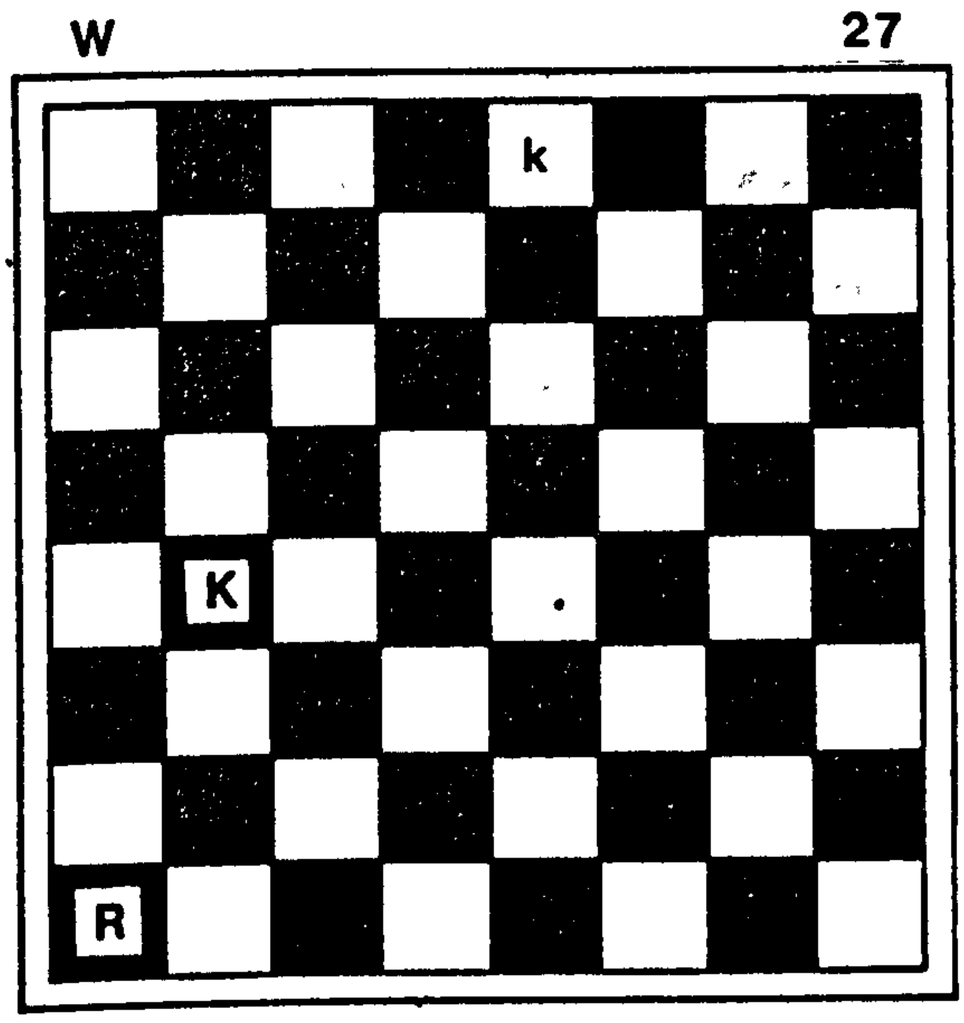
(i) the number of ranks or files to which the Black King is confined by the Rook, whichever is the smaller (function 1);

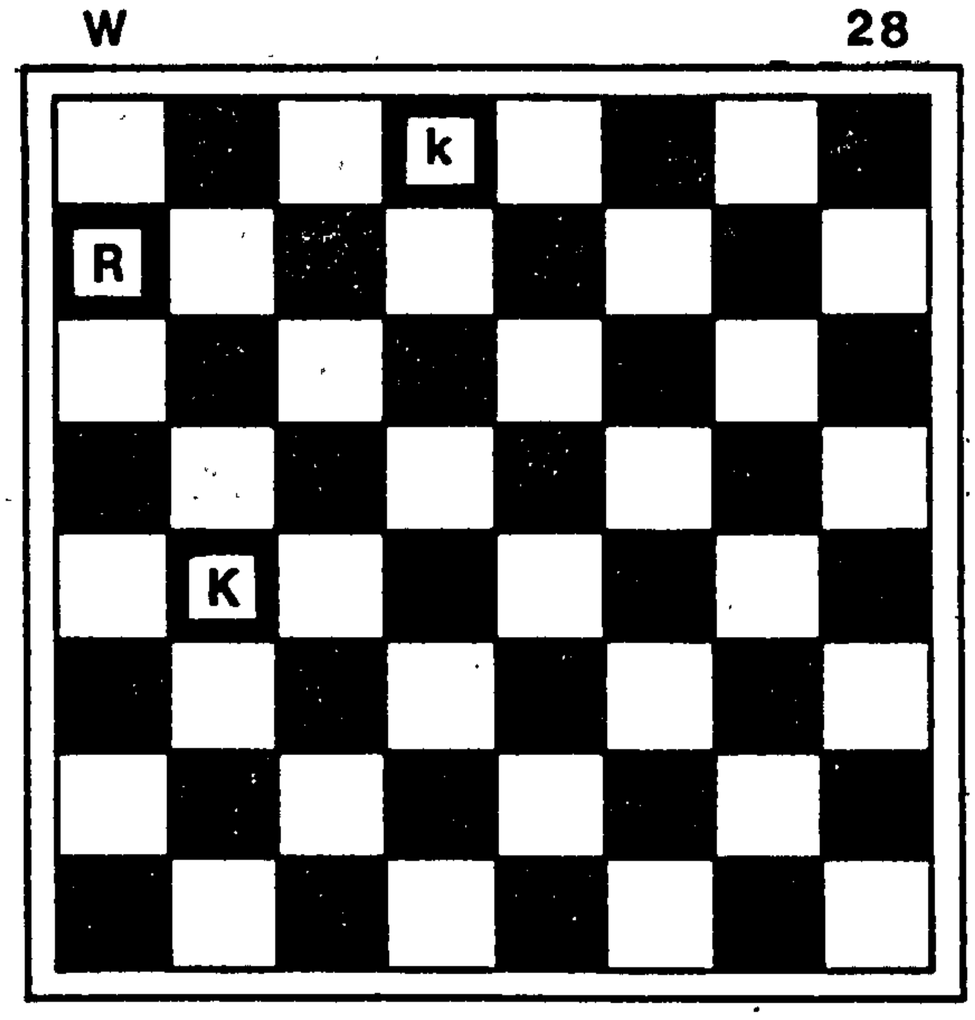
(ii) the block distance between the two Kings, i.e. the file or rank difference between the Kings, whichever is the larger (function 4); and

(iii) the file or rank difference between the Kings, whichever is the smaller (function 5).

(Note that (ii) is only used if (i) is insufficient to discriminate between the positions and (iii) is only used if both (i) and (ii) are insufficient.)

Using the two functions 4 and 5 in the above way is a means of making precise the idea "minimize the distance between the Kings".





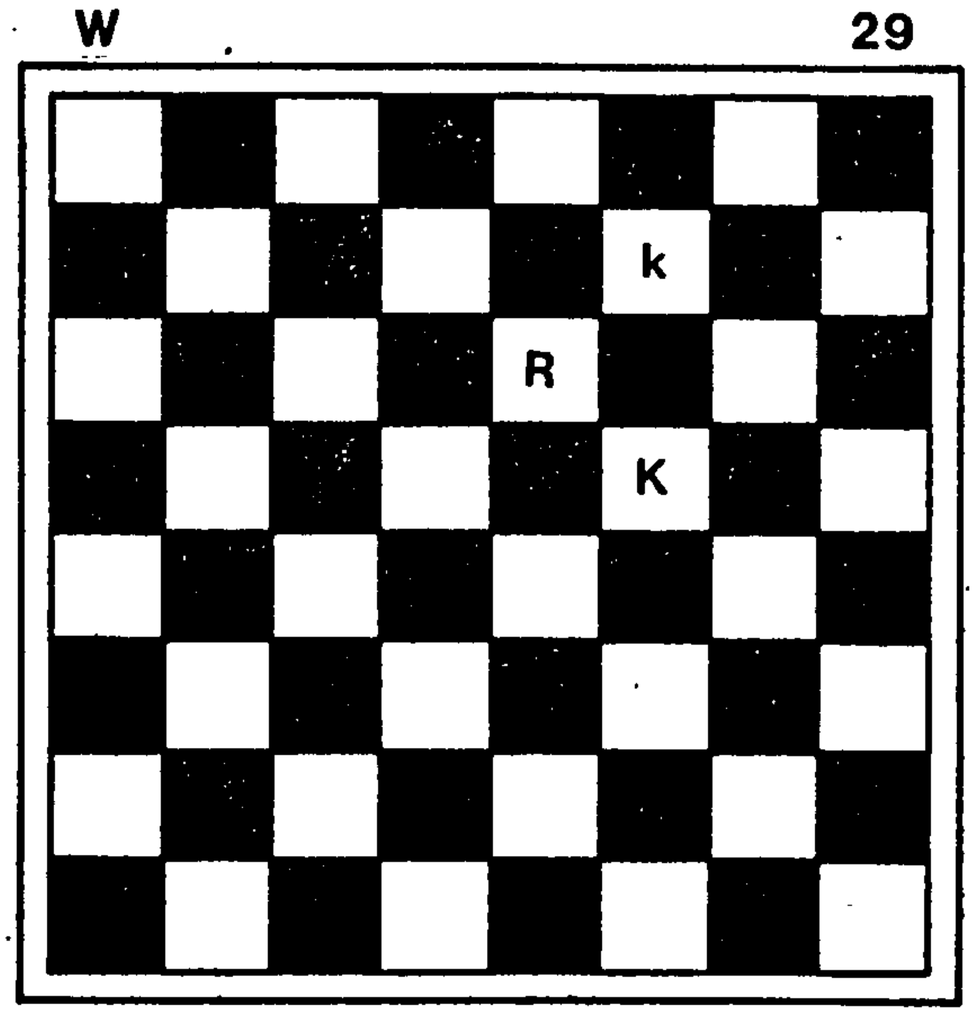
In Figure 27, White plays R-A7 restricting Black to only one rank at the top of the board.

In Figure 28, White must choose between King moves to A3, A4, A5, B3, B5, C3, C4 and C5, any of which would give a position in class 10 with a value of 7 for function 1 (since Black is restricted to one rank). Of these squares, A5, B5, and C5 all give equal values of the smallest block distance (i.e. 3). Thus function 5 is finally used to distinguish between these three remaining possibilities and K-C5 is chosen, giving a value of 7 for function 5.

4.5 Example

As a detailed example of the use of class values and the associated functions in the move-finding algorithm, consider the position given as Figure 29 below.

W 29



White's problem here is to make progress towards the checkmating position while preventing Black's King from "escaping" to the other side of the board.

If White plays R-F6ch or R-G6 etc., then Black simply plays K-E7. If White instead plays R-E5, then Black plays K-G7 and White can do no better than move his Rook back to E6 allowing Black to repeat Figure 29 by K-F7.

The winning manoeuvre is 1.K-E5, and if Black replies K-G7, then 2.R-F6 confining the King to a quadrant of only four squares.

With this discussion in mind, a table of legal White moves from the position given in Figure 29 is set out below, together with the class and position value of the successor position corresponding to each move.

Table 4 Initial algorithm: an example of move selection

Move	Class of successor	Position value of successor	Comments
K-G5	1	2000000	Rook en prise
K-G4	1	2000000	Rook en prise
K-F4	1	2000000	Rook en prise
K-E4	1	2000000	Rook en prise
K-E5	7	7440000	Squad is 6, so value of function 2 is 44
R-E7	1	2000000	Rook en prise
R-E8	1	2000000	Rook en prise
R-F6	11	3000000	"Residual" position
R-G6	9	5380800	Value of function 2 (50-squad)=38 — Kings on same file, so value of function 3 is 8.
R-H6	9	5360800	Value of function 2 is 36 — Kings on same file, so value of function 3 is 8.
R-E5	8	6000000	The Kings are in vertical opposition with the Rook one file closer to the centre - with no associated functions.
R-E4	8	6000000	
R-E3	8	6000000	
R-E2	8	6000000	
R-E1	8	6000000	
R-D6	9	5420800	50-squad is 42 — Kings on same file so value of function 3 is 8.
R-C6	9	5400800	50-squad is 40
R-B6	9	5380800	50-squad is 38
R-A6	9	5360800	50-squad is 36

White's move K-E5 leads to a position in the highest ranked class (class 7) and is accordingly chosen as best.

4.6 Discussion

In terms of the objectives set out in Section 1, this section has demonstrated that, at least for a relatively simple endgame, an algorithm can be constructed using the overall framework of the model described in Section 2, which relates directly to the descriptions given in standard textbooks.

Moreover, it is reasonable to say that the complexity of the algorithm is commensurate with the apparent complexity of the problem. Mapping a position with Black to move to one of the eleven equivalence classes requires a maximum of ten tests and only one assignment. Finding the value of each associated function required involves only a simple arithmetical calculation. To find White's move in any given position, this process needs to be carried out once for each possible successor position (22 at the most) and the most favourable position value chosen. The total amount of computation required is therefore very small and an initial heuristic pruning of White moves would reduce this still further. The algorithm given makes no use of tree-searching which fits in well with the observations made previously that the strong player's ability in endgame play is based far more on an extensive knowledge of significant patterns than on powers of deep analysis and that textbook descriptions of King and Rook against King make little or no mention of the need to perform detailed analysis. As was pointed out at the beginning of the section, the form of the algorithm given here was arrived at by an iterative process of successive refinement.

A small test file of positions derived from a number of textbooks was used to monitor the performance of the program at each stage. Since checkmate,stalemate and "Rook en prise" positions have been identified individually as equivalence classes, with suitable class values, it is safe to say that the program will never miss a checkmate and will never give stalemate or leave its Rook en prise, since it can easily be proved that there are always other better alternatives available.

The inclusion of class 4 ensures that a checkmate in two moves is also never missed.

The general strategy is embodied in rules 5 to 10, with class 11 intended to contain only unimportant and insignificant positions. Although it is expected that positions in classes 1,3 and 11 will never be selected as best by the program, the first two play an important part in enabling the program to avoid disastrous combinations and the third ensures that every position must belong to some class and thus ensures that the algorithm will always produce some valid move in any position irrespective of the choice of the other classes at any stage of the iterative process.

The exact ordering of the rules given in the algorithm is to some extent arbitrary and has been chosen to simplify the definitions where possible (for example, by testing first whether the Rook is en prise). By contrast, the order of class values is intended to be significant and to reflect the relative importance of each of the features of a position concerned.

Having demonstrated that the model can be used in this case to produce a reasonably good algorithm, what remains is to demonstrate that the model is also applicable to other endgames, to investigate how readily such algorithms can be improved in the light of testing and to examine the possibility of mechanizing this process.

These aims are addressed in the following sections.

5. Testing the algorithm: I - some problems of testing correctness

The discussion given in the previous section provides some justification for the choice of algorithm. However, the question naturally arises, how could a program embodying this algorithm (or any other similar program) be proved to be correct? The discussion here is confined to programs for King and Rook against King but the argument is equally applicable to other chess playing programs.

Any King and Rook against King position with White to move is a theoretical win for White and any move which does not give stalemate or leave the Rook en prise maintains the win. Nevertheless, it is clear that White cannot win simply by playing a series of win-preserving moves; he must make progress in some way, towards a checkmate position. Four qualitative levels of program can be identified in the case of King and Rook against King:

(i) optimal programs, where in any given position White always chooses the move leading to the most immediate checkmate against Black's best defence (the minimal path move);

(ii) sub-optimal programs, where White always wins, but not always in the smallest possible number of moves;

(iii) programs where White always plays win-preserving moves but fails to achieve checkmate from some positions;

(iv) programs where in some positions White stalemates Black or leaves his Rook en prise.

In the general case, these four levels can be classified as

(i) optimal,

(ii) sub-optimal (making progress towards the most favourable outcome from every position),

(iii) result-preserving but failing to make progress from at least one position,

(iv) incorrect (failing to maintain the most favourable result in at least one position).

Programs at levels (i) and (ii) can be considered correct, those at levels (iii) and (iv) are incorrect. The program described in Section 4 would appear to belong either to level (i), level (ii) or possibly level (iii). It certainly does not belong to level (iv), since a move which gives stalemate or leaves the Rook en prise will never be chosen, as noted in the previous section.

A conceptually simple approach to proving a program optimal (level (i)) would be to establish a look-up table containing the minimal path move in every legal position with White to move (assuming that these moves were known by some external means). To test whether a given program belonged to level (i), it would then be necessary to generate its move in each legal position in turn and compare these moves with those in the table.

Even for a relatively simple endgame, this kind of exhaustive testing would be a substantial computational task. An additional difficulty is presented by the fact that there are a large number of positions in which White has a choice of equally good "best" (i.e. minimal path) moves and hence there are a correspondingly large number of equally good

"optimal" programs belonging to level (i). In order for each of these programs to be verified and found correct, it is necessary for the look-up table to contain not only one, but all of the minimal path moves in each position.

Even then it would not be possible to determine whether a program which occasionally played moves not present in the table belonged to level (ii) or level (iii), that is to distinguish between correct (but not perfect) play and incorrect play.

Clarke (1975) has set up databases which enable all the minimal path moves in each legal position to be calculated for the two endgames King and Rook against King, and King and Pawn against King, and some experimentation based on the latter is described in the second part of this report.

However, to test an algorithm fully using even such relatively small databases is a substantial project. For more complex endgames, the number of positions to be stored will generally be much larger and the time taken both to set up the database and to validate an algorithm using it is likely to become prohibitive.

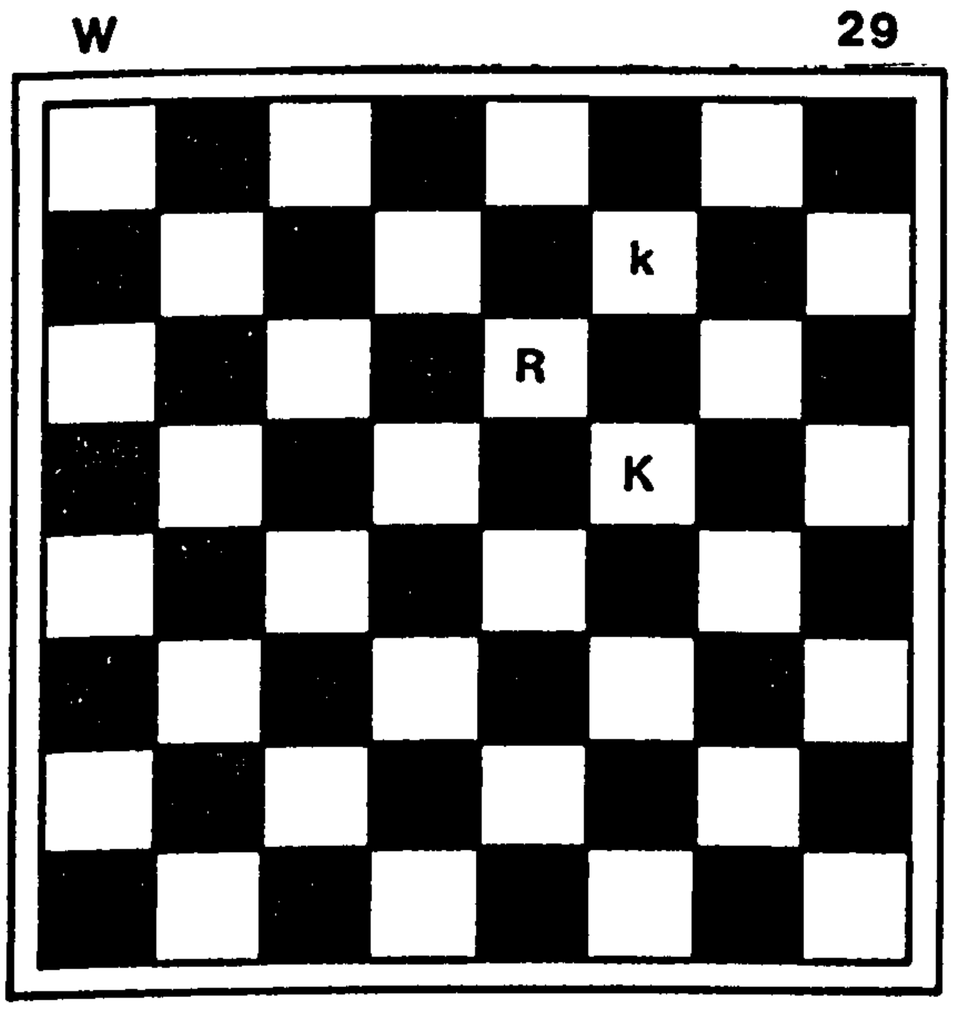
If the possibility of automatic proof of the absolute correctness of a program has in general to be discounted, then the evaluation of a program must inevitably be based on the subjective opinions of expert chessplayers.

This too presents a number of difficulties.

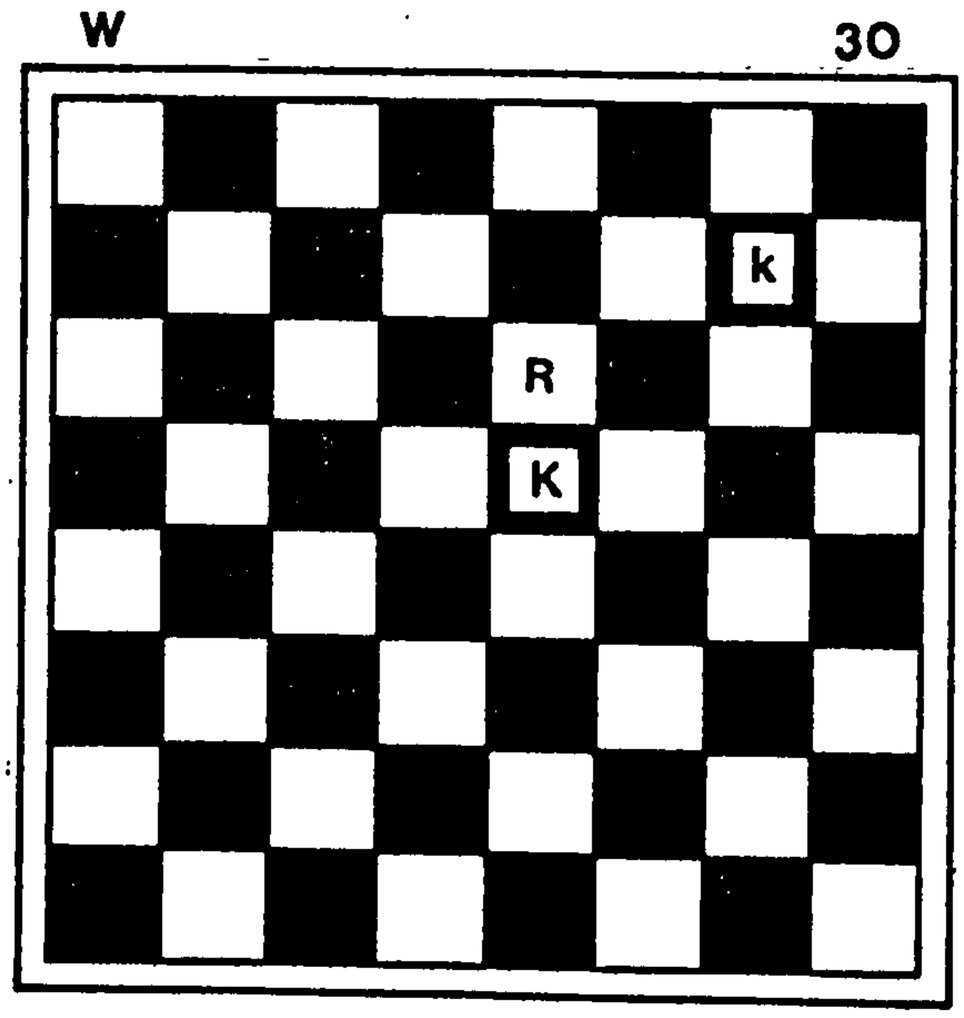
It is virtually inconceivable that even the most expert of human players could infallibly determine whether a given program belonged to level (i) or level (ii), even for King and Rook against King. (The discussion of Figure 51 in Section 7 is relevant here.)

To take an extreme case, it may be necessary to decide that a move generated by the program (which mates in 16 moves, say, against best play) is slightly inferior to a single alternative, which leads to mate in 15. Such judgements are probably too fine even for expert chessplayers and are, of course, of no importance in actual play.

Identifying a program as belonging to level (iv) is a relatively straightforward problem of searching for an example of a drawing move made by White, but determining whether a program belongs to level (ii) or level (iii) is more complex. Consider, for example, Figure 29 (which is repeated below for convenience).



29



30

As was shown in Section 4.5, the move played by the program in this position is K-E5. If Black now plays K-G7, the resulting position

is given in Figure 30. The move the program chooses in this position is now, in fact, R-F6, restricting Black to a quadrant of only four squares.

There is no doubt, however, that any White move in this position will win (except R-G6 or R-H6). Suppose that for some program the moves played are K-E5 in Figure 29 but K-F5 in Figure 30. After this second move, White still has a forced win against any play by Black, but it is Black's move and if he plays K-F7 the position is now identical to Figure 29. White would again play K-E5 and Black could choose to reply K-G7 returning to Figure 30 and the sequence would continue as before indefinitely.

What has gone wrong in this situation is not that White has made any particularly bad moves (he has a forced win at every stage!), but that he has failed to make any progress.

As long as Black does not vary the sequence, White will cycle continuously between Figure 29 and Figure 30. (Whether or not the three-fold repetition of position drawing rule is implemented is irrelevant here. The program would evidently have failed.) Although this is a fairly simple example, it would be possible to create examples of longer cycles of seemingly plausible moves. This phenomenon of cycling is, in fact, a commonly occurring problem for this kind of endgame. Cycling and "failing to make progress", in general, produce difficulties when attempting to judge any move-finding algorithm. Each individual move in a sequence may be good but the result of the complete sequence may be that no progress has

been made. (Human chessplayers also find cycling a problem, but are sufficiently flexible deliberately to vary the sequence to break out of the cycle.)

To determine whether a given program belongs to level (ii) or level (iii) the human chessplayer must be able to detect such cycles. This can only be achieved by looking not at individual moves in isolation but at sequences of moves, with a corresponding increase in the complexity of testing.

In practice, it would seem that to prove the absolute correctness of a program is prohibitively difficult and the best that can be achieved is some approximation to this ideal. A variety of different empirical methods of program testing are employed in the sections which follow.

6. Testing the algorithm: II - empirical investigations

As an experiment, a program embodying the King and Rook against King algorithm described in Section 4 was made available to Open University students at Mathematics Faculty Summer Schools in July-September 1975. Students were allowed to set up an initial position, either of their own choosing or generated at random by the program, and to play on from that position until either they were checkmated or they terminated the game. The program had the Rook and made the first move in every case. Students were invited to report any interesting occurrences and all games played were automatically recorded on file for subsequent testing and inspection. The exercise was repeated a few months later at two "open days" attended by members of the council of the Open University and the general public.

The purpose of this testing was essentially twofold. Firstly, as an empirical test of the algorithm described in Section 4. Although the algorithm was created primarily as an example of the means by which information derived from textbook descriptions and examples could be incorporated into an algorithm within the framework of the model described in Section 2, it was hoped that the algorithm would be correct (in the sense of playing well by human standards) in the great majority of cases.

As was pointed out in Section 4, in creating any algorithm of this kind it is inevitable that some errors, either major or minor, will be made and the second objective of this testing was to examine how easily any errors found in the program's play could be corrected by changes to the algorithm.

One important criterion in the evaluation of the model was that it should be possible to make such corrections with only relatively minor changes to the algorithm. The possibility and the desirability of improving a program to play perfectly in all positions will be considered in later sections. At this stage the principal concern was to check and, if necessary, to improve the program's play such that it performed well, by human standards, in every position with which it was presented.

Several hundred games were played against the program, by players of all levels of ability from beginners to national championship standard, comprising a total of 2486 positions with the program (White) to move. A fairly large number of examples of poor play by the program were identified, either by the human opponents themselves or by subsequent analysis of the games played.

However, the changes needed to the algorithm to remedy these problems turned out to be very small, as a closer inspection revealed that almost all of the difficult positions could be grouped together in a way which related directly to the equivalence classes used in the original algorithm. A number of small changes were made to the definitions of the equivalence classes or their associated functions and three new classes were added to cope with some particularly difficult situations and to produce significantly stronger play in others.

A description of the problems encountered and the corresponding changes made is given below, followed by a summary of all the changes made to the original algorithm, the revised algorithm in tabular form and a description of the basic primitives of which the revised rules and associated functions are composed.

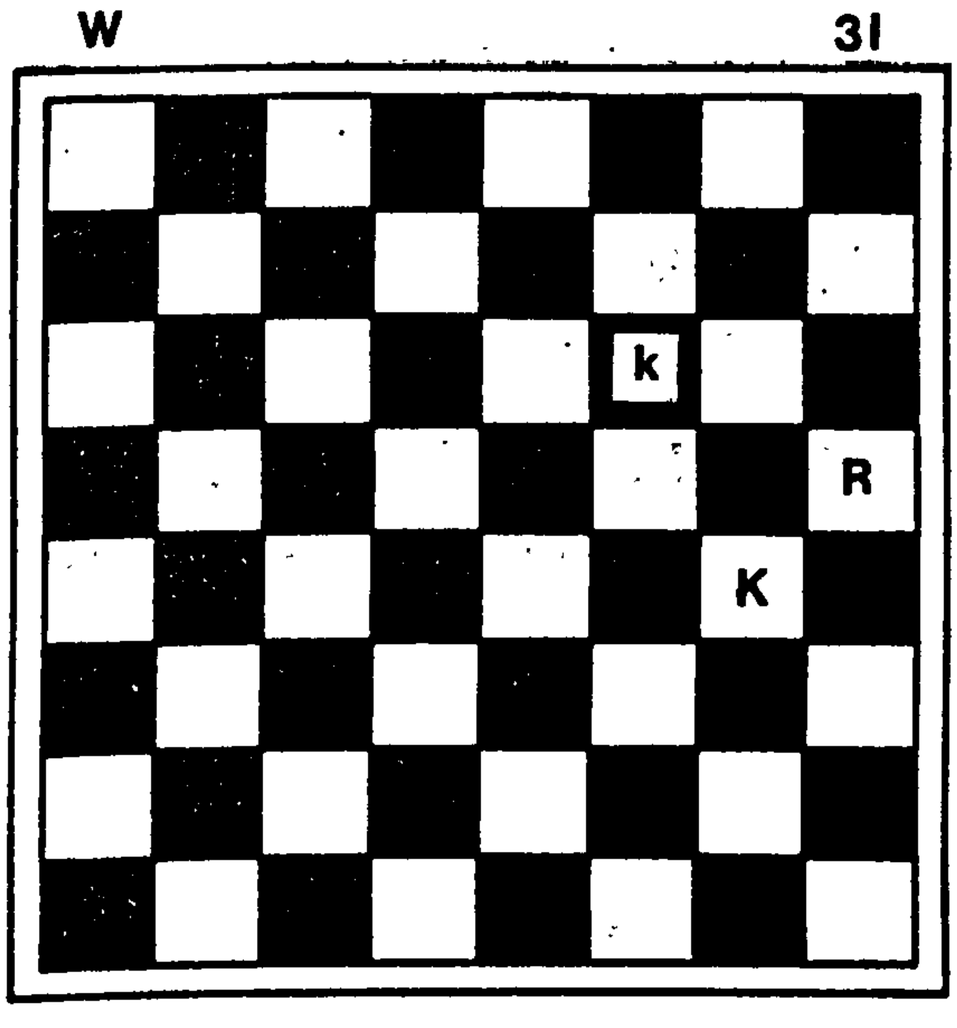
The changes made to the original algorithm and the revised version are discussed further in Section 7.

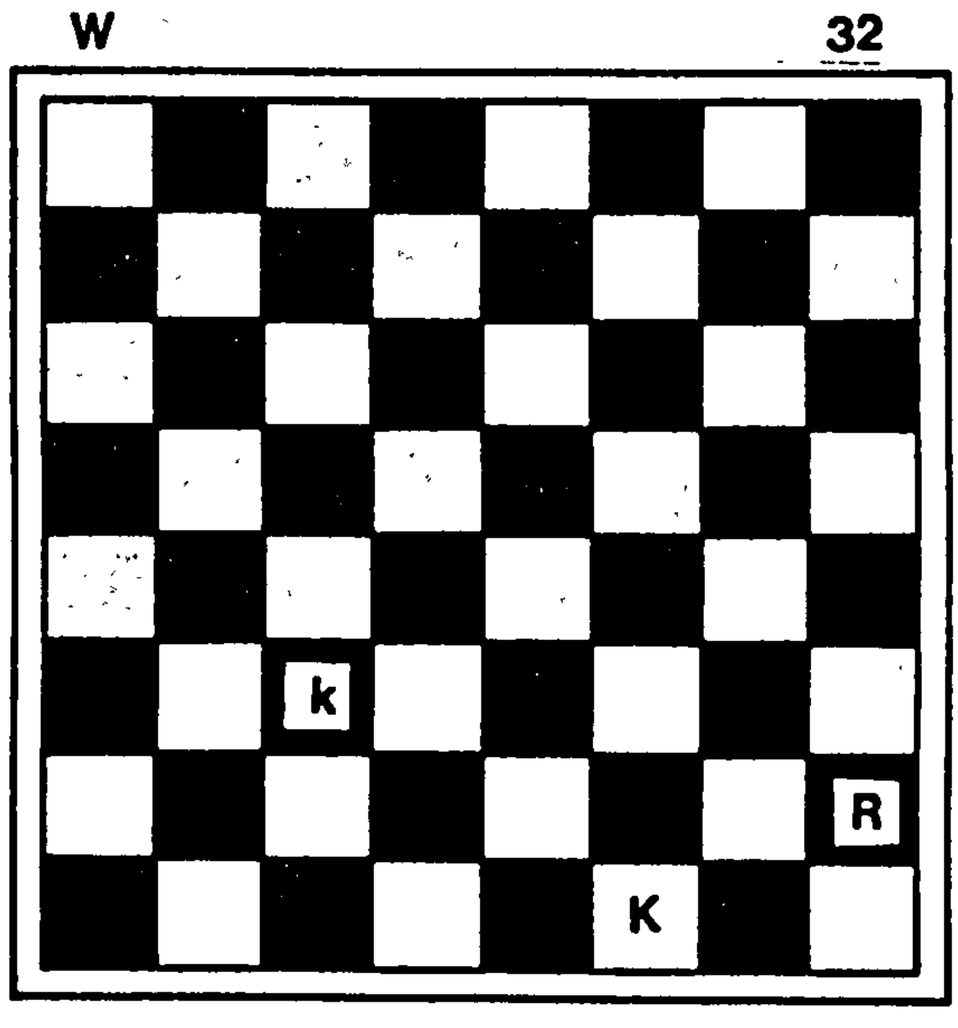
6.1 The problems identified and the solutions implemented

(1) Specifying a small quadrant

One principle underlying the definition of class 9 is that Black should be restricted to a small quadrant of the board. However, it became clear that the definition of rule 9 was insufficient to ensure this, as will be seen from the following examples.

Figure 31 is typical of a large number of instances in which the program played a poor move.





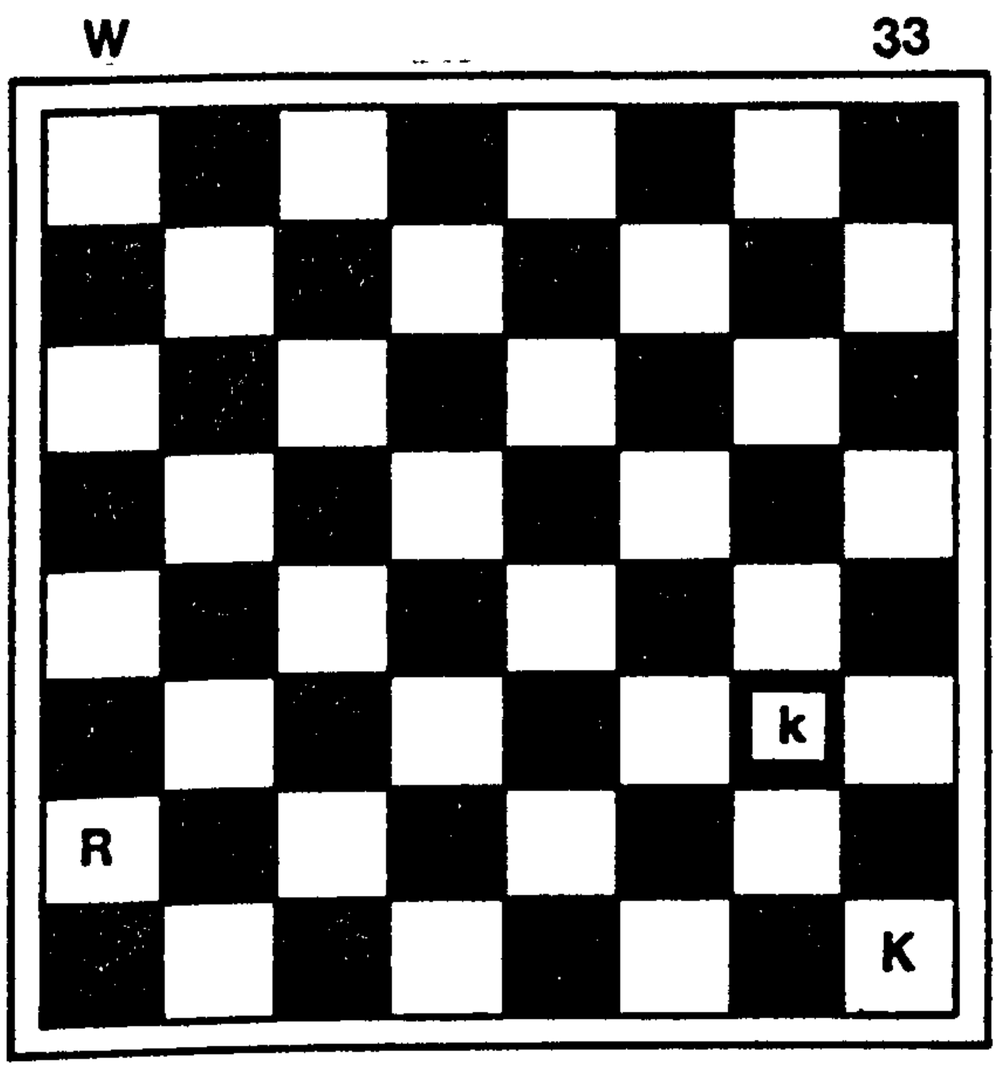
The game continued from this position

1. R-G5, K-E6; 2. R-F5.

Both White's moves here are poor. They are played because the resulting positions are classified as members of class 9 and therefore better than the positions arising after the alternative K-F4 on

either White's first or second move (both class 10). The principle behind the definition of class 9 is that Black should be restricted to a small quadrant of the board, but after 1. R-G5, for example, the quadrant consists of as many as six files.

In Figure 32, White played 1. R-E2, again resulting in a Class 9 position, this time with Black restricted to six ranks of the board, the superior 1. R-H4 resulting only in a (lower ranked) class 10 position.



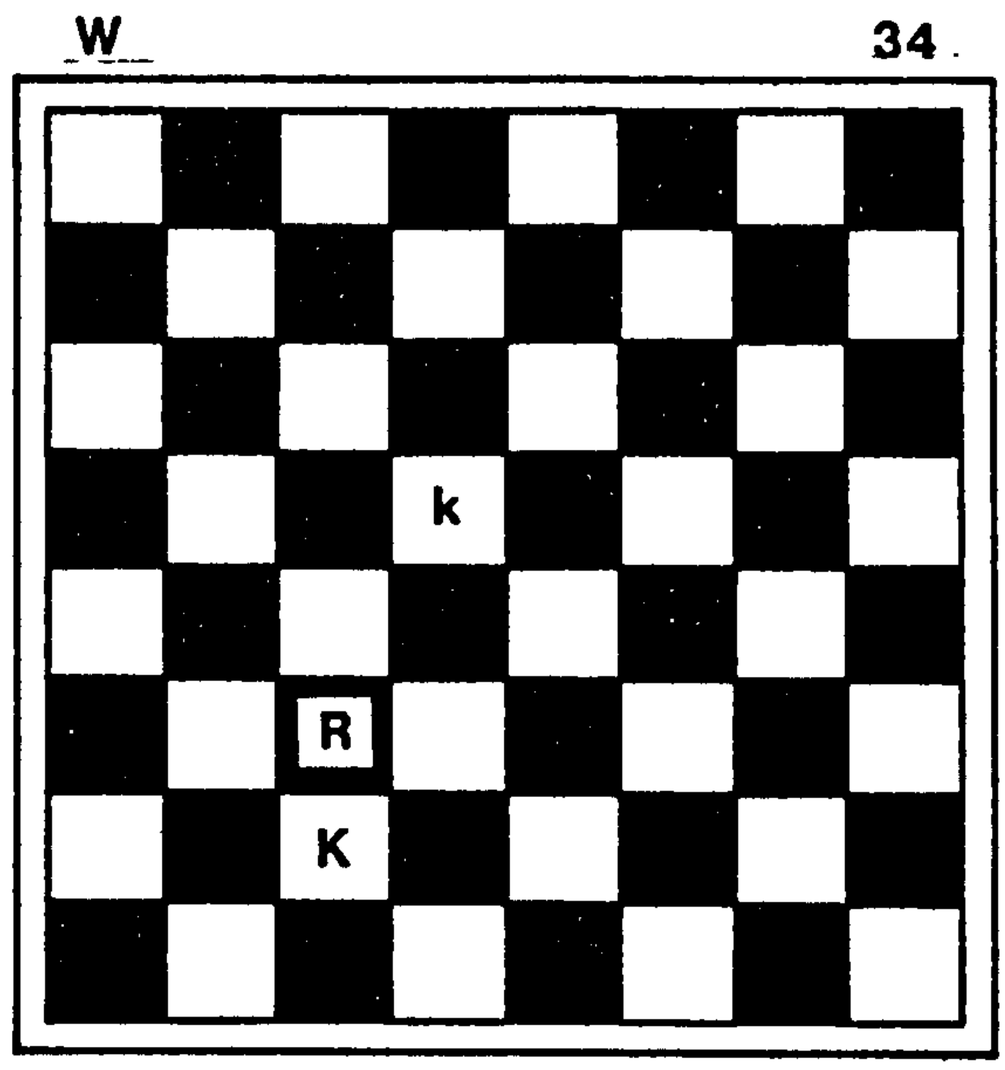


Figure 33 shows an extreme case where after 1. R-H2 Black is "restricted" to no fewer than 42 squares of the board!

In all of these examples, the program merely played poorly but nevertheless won in the end. However, there were a few instances in which it even led to a draw. Thus in Figure 34 play continued

1. K-B3, K-E4; 2. K-C2 (White's move in each case giving a Class 9 position) and Black then returned to the original position by playing K-D5, with a repetition of moves following. In this case the program's play was not merely poor but incorrect, in the sense of failing to lead to a win. To improve the program's play the definition of Class 9 was altered to exclude the kind of position arising in the above variations. The additional restrictions were made that a position should only be included in Class 9 if Black was restricted to a quadrant of no more than four ranks and four files. With the orientation adopted for the definition of this class, these conditions can be written as

$$(WR1 \leq 5) \text{ AND } (BK2 \geq 5).$$

With this change, taking the example of Figure 31, the positions after 1. R-G5 and 1. K-F4 are now both members of Class 10 and will be decided between on the basis of the functions associated with that class (and in fact 1. K-F4 will be chosen).

(2) Restricting Black to the top half of the board

Following the change described above to the definition of rule 9, it became clear that the definitions of rules 5 to 8 also needed refining. In each case the White King is (by definition) two ranks below the Black and it is White's aim steadily to force Black back towards the top of the board. This strategy is unlikely to be the most effective way of winning unless Black is already in the top half of the board.

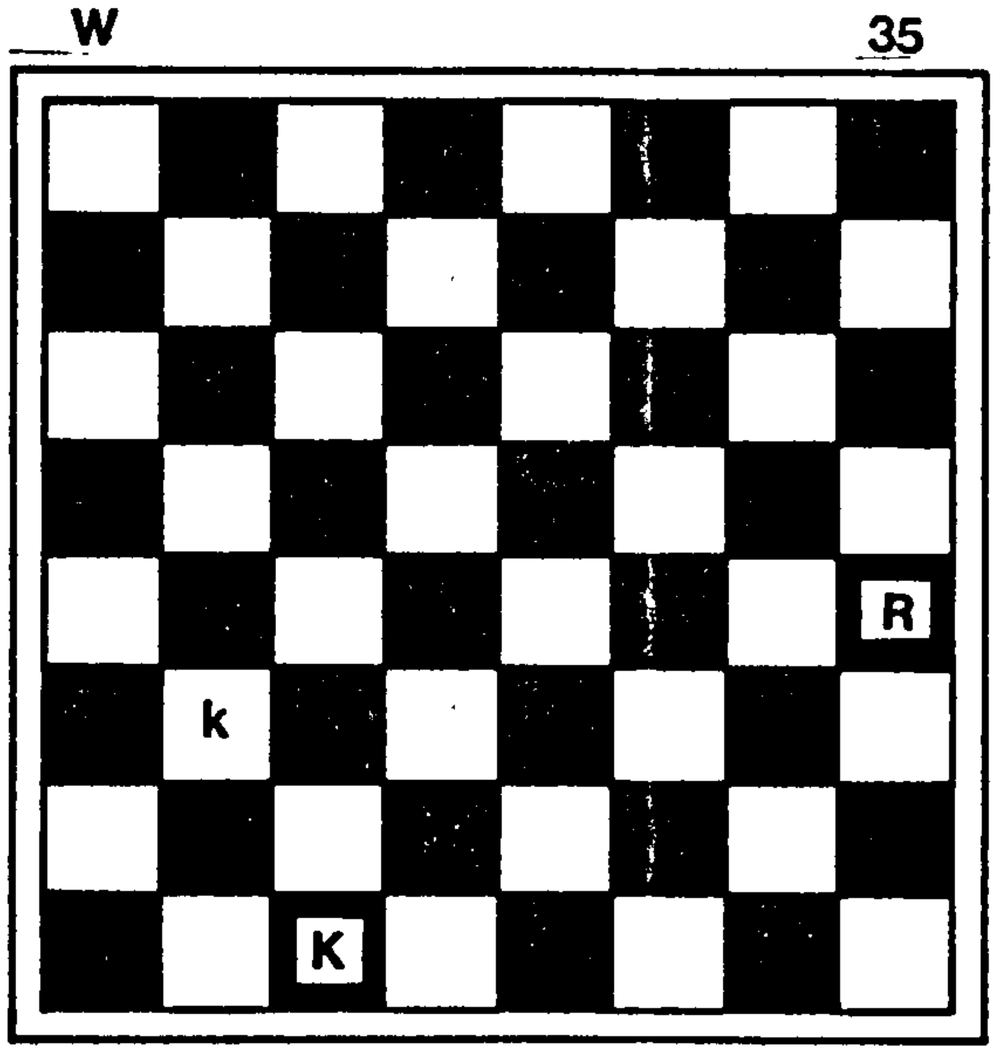


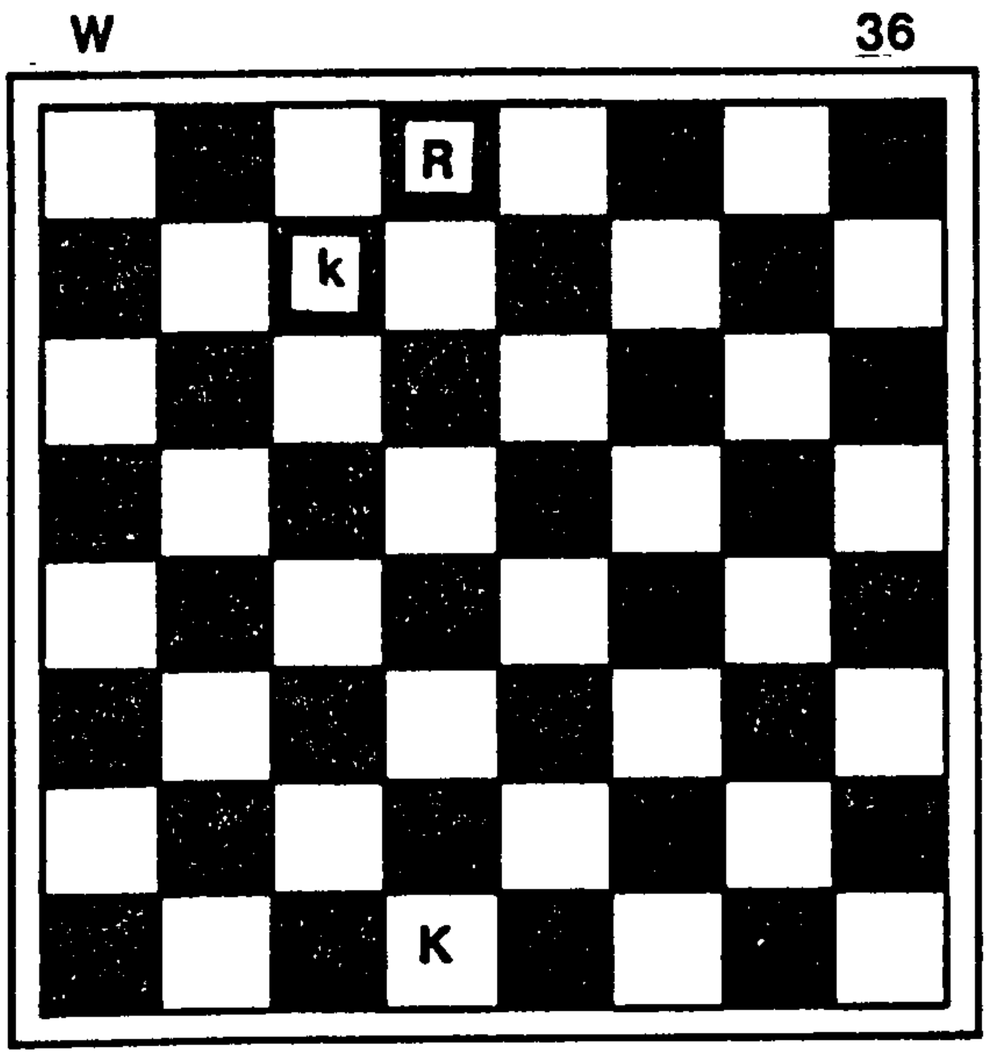
Figure 35 shows a typical example. Here White played the weak move 1. R-H2 (giving a position in class 7), with the Black King "restricted" to the top six ranks of the board. To avoid such situations, an overall change was made to the definitions of classes 5 to 8, the following additional condition being added

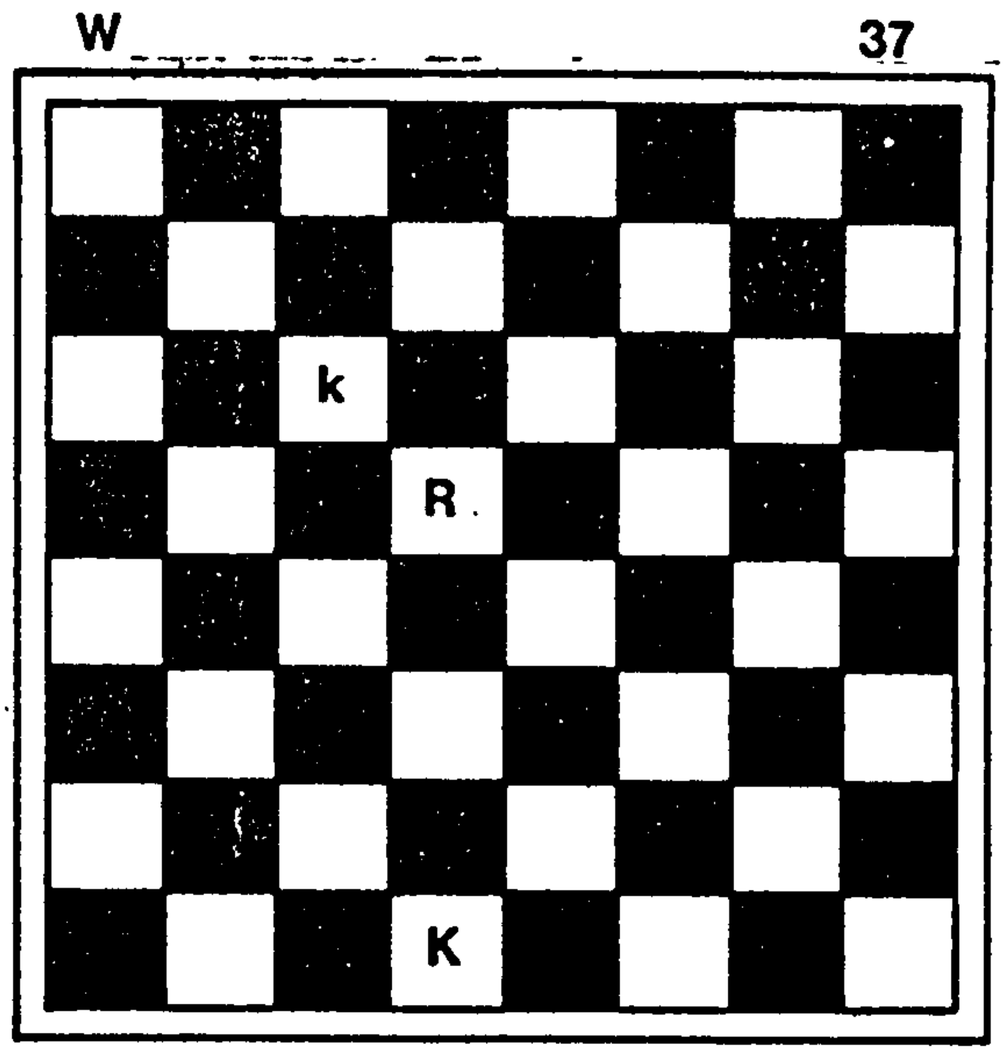
$$BK2 \geq 5.$$

Figure 35 is, in fact, typical of a number of positions which it is difficult to handle efficiently by the "chasing" process alone. Such positions are discussed in (5) below where a different method of play is introduced. With the changes given there, White's move in Figure 35 will be 1. K-D2.

(3) Keeping the Rook at a distance

The three associated functions for class 10 proved to be insufficient in some situations and a fourth one was added.





A draw by repetition of position occurred in the play arising from Figure 36.

Here White had to choose between moving his Rook to D5, D4, D3 and D2. In each case the Black King is restricted to only 3 files (giving the largest available value of function 1), and the Kings are the same distance apart.

Thus the values of functions 1,4 and 5 were the same and the program's move was chosen 'arbitrarily' by selecting the first of the tied 'best' moves to be generated (by the legal move generator) as the one to be played. Rook moves are generated in the program in the following order:

(i) moves forward along the file,

(ii) moves to the right along the rank,

(iii) moves backward along the file,

(iv) moves to the left along the rank.

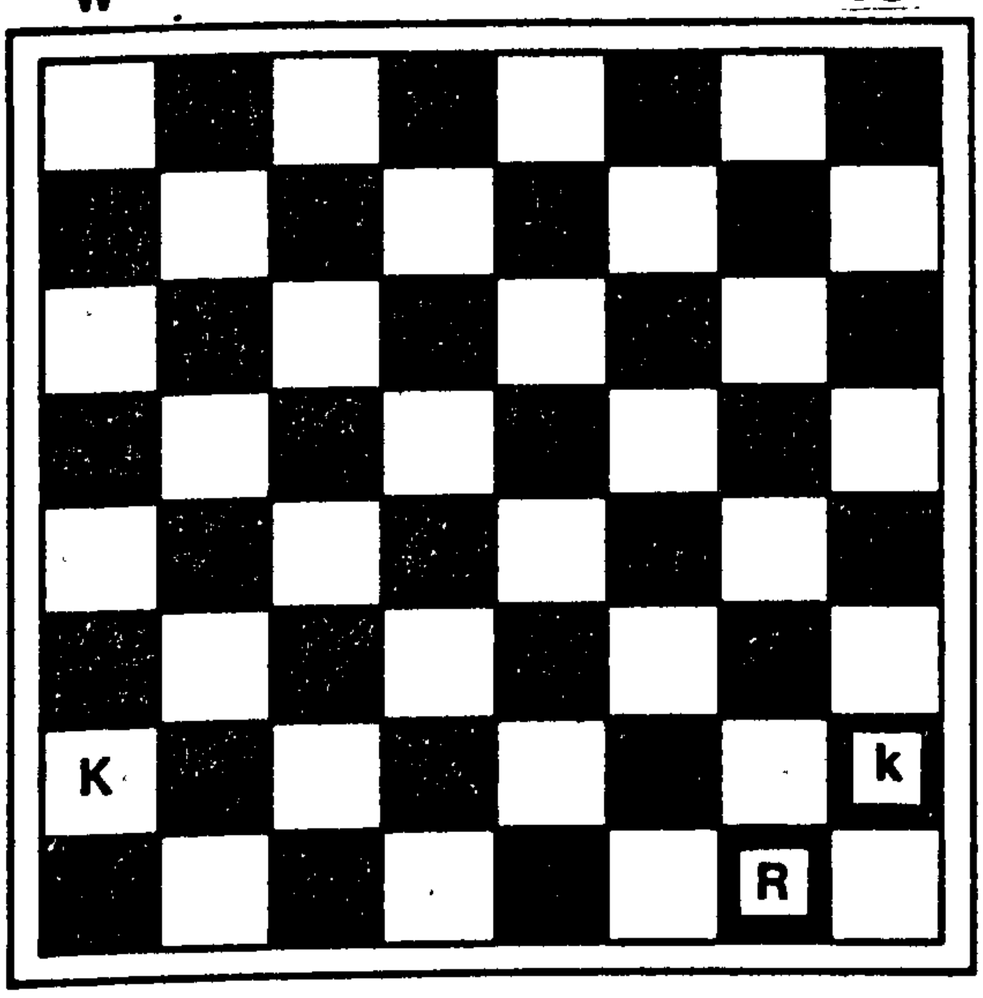
In each stage, moves of the smallest number of steps from the original position are generated first.

In this case the first of the four Rook moves to be generated was R-D5 and this was accordingly White's move. Black now played K-C6 giving Figure 37.

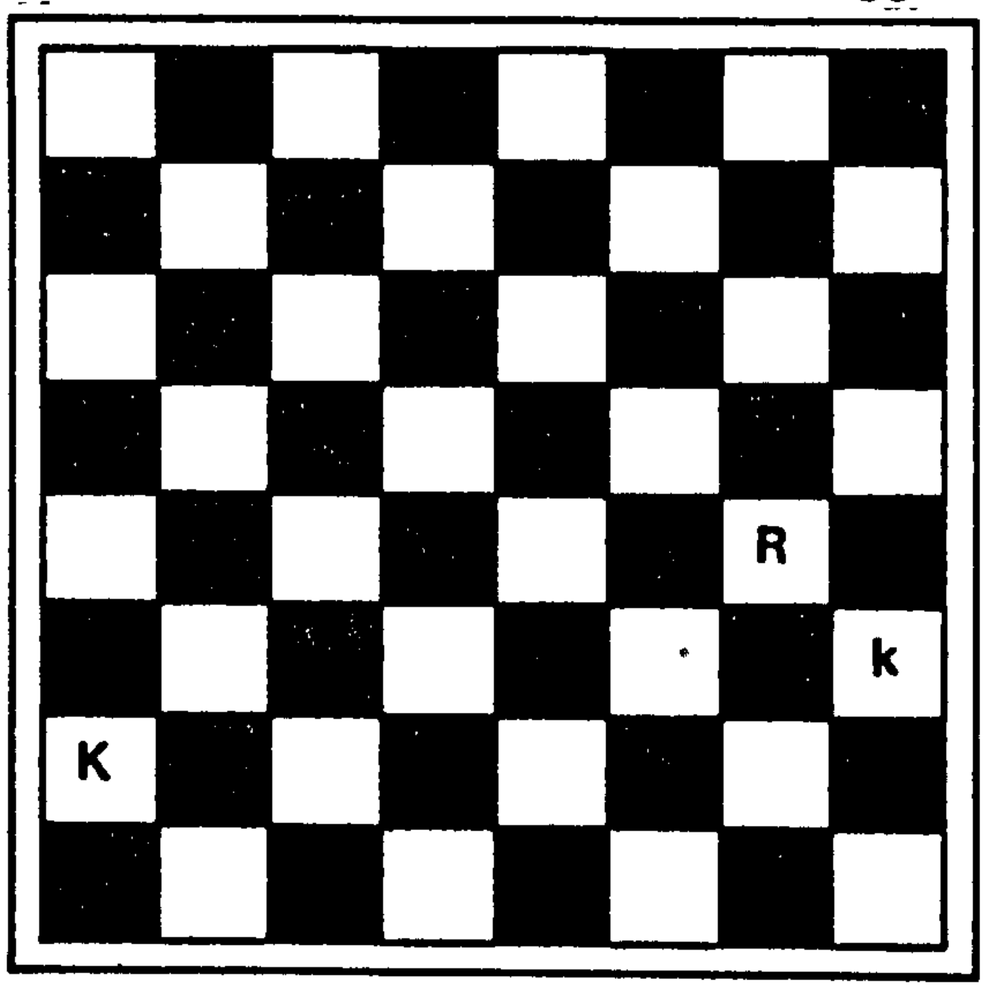
Again White had to choose amongst four Rook moves, to D8, D4, D3 and D2, which led to tied Class 10 positions. In this case the first generated was R-D8 and this move was played.

Black now replied K-C7, returning to the original position, with a repetition of position to follow.

W 38



W 39



A longer sequence of moves leading to repetition of position occurred in the play from Figure 38, which continued:

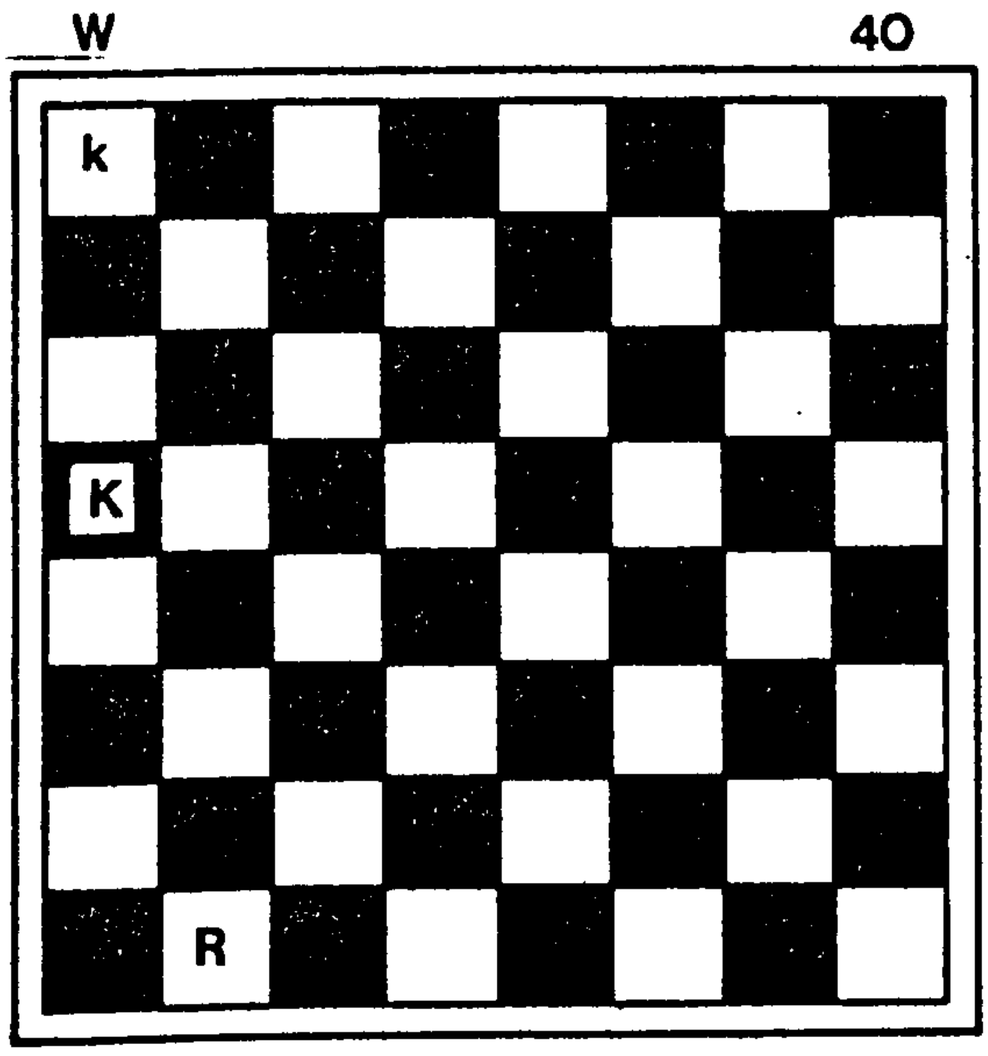
1. R-G4, K-H3; 2. R-G5, K-H4; 3. R-G6, K-H5;

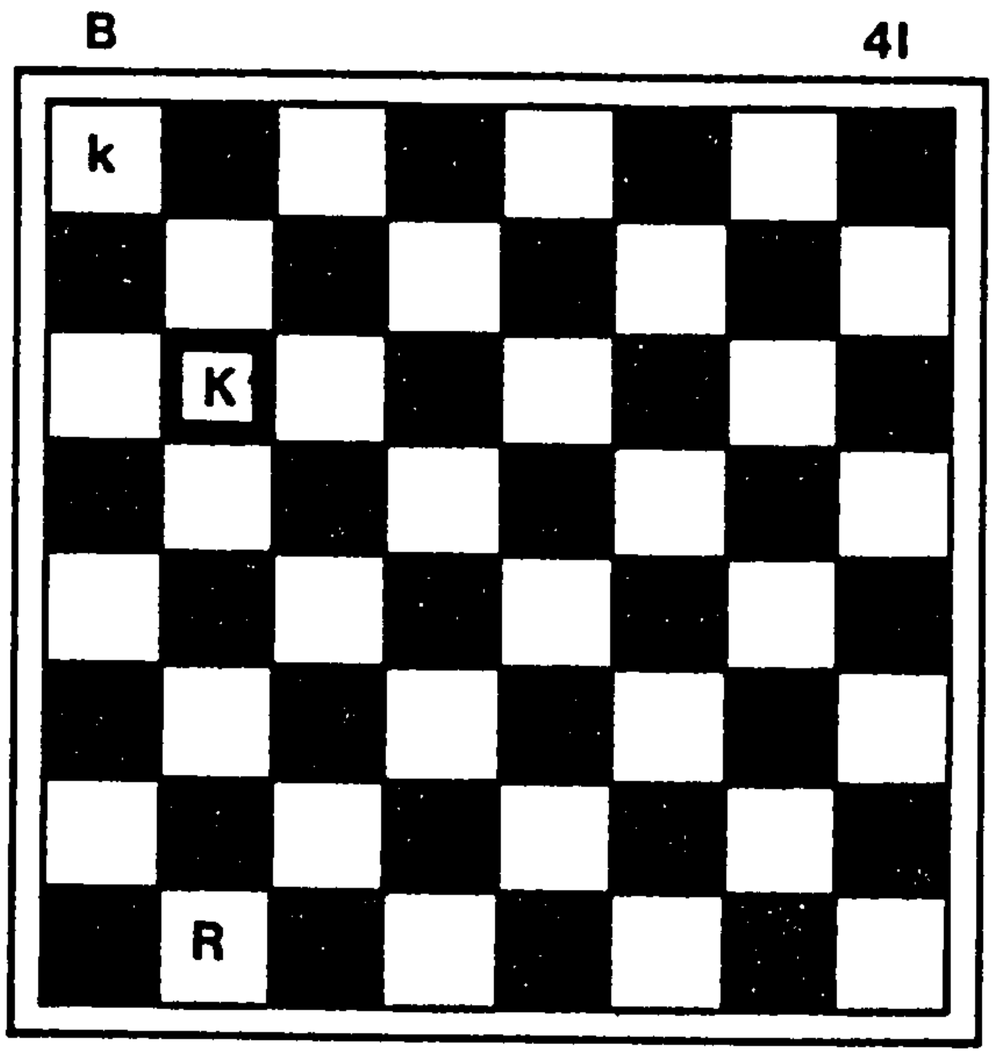
4. R-G7, K-H6; 5. R-G8, K-H7; 6. R-G5, K-H6;

7. R-G8, K-H7 etc.

One possible solution to this problem which would avoid the need to introduce a further associated function for Class 10 would be to choose in the case of a tie not the first generated but the last generated of the tied moves. However this solution, although avoiding repetition of position, still leads to weak play in some positions. For example, in Figure 39 the program will then play the weak first move 1. R-G1, although after Black's reply K-H2 it will choose the correct second move R-G8 and thus avoid the repetition of position which occurred above. More fundamentally, since any method of resolving ties is essentially arbitrary, such a method should be used only when the choice is genuinely arbitrary and not, as in this case, when one of the "tied" successor positions is much more favourable than another. Since the difficulty here arose in connection with resolving ties between successor positions in Class 10, the solution adopted was to introduce a fourth associated function, with value the block distance between the Black King and the Rook, i.e. dist (Black King, Rook), rather than changing the definition of rule 10 or any of the other three functions. With this change, the program plays correctly in the positions given, for example in Figure 39 the first move is 1. R-G8. Note that with this change it is also necessary to associate a fourth (null) function with each of the other equivalence classes.

(4) The Black King in the corner





Difficulties arose in connection with stalemate possibilities in a small number of cases similar to Figure 40. (The same effect would occur with the Rook on any one of the squares B1, B2, B3, B4 or B5, and the positions symmetrically equivalent to these.)

In this position White's 'natural' move K-A6 gives stalemate and play continued instead

1. R-B2 (class 10) K-A7
2. R-B6 (class 6) K-A8

followed by a further Rook move on the B - file by White, then K-A7 by Black, whereupon White played R-B6 returning to the position after his second move, with an eventual draw by three - fold repetition of position. This problem only occurs with the Kings on the Rook file as in Figure 40 and is a consequence of the stalemate rule which, in a sense, introduces a 'discontinuity' into White's strategy - it is necessary to restrict Black as much as possible but not to a

position where he has no legal moves at all.

White's best continuation in Figure 40 is, in fact, to play 1. K-B6 with checkmate in three moves, after

1. ..... K-B8
2. R-C1 K-A8
3. R-C8 Mate

The simplest way to handle the complications which arise from Figure 40 would seem to be to recognize them as a special case which occur in a small number of situations only and to deal with them accordingly.

In general, when making corrections and adjustments to an algorithm it is important not to introduce "special cases" indiscriminately but in this case there is a clear reason for their occurrence - stalemate possibilities with the Black King in the corner of the board. The easiest solution is to introduce a further equivalence class, number 12, which consists of positions such as Figure 41, with Black to move. Since from these positions White mates in two moves whatever Black plays, the class is ranked between the existing classes 4 and 5 and is tested for after rule 4. The corresponding rule is as follows (assuming the orientation already pertaining after rule 4):

BK1 = 1 AND BK2 = 8 AND WK1 = 2 AND WK2 = 6 AND WR1 = 2.

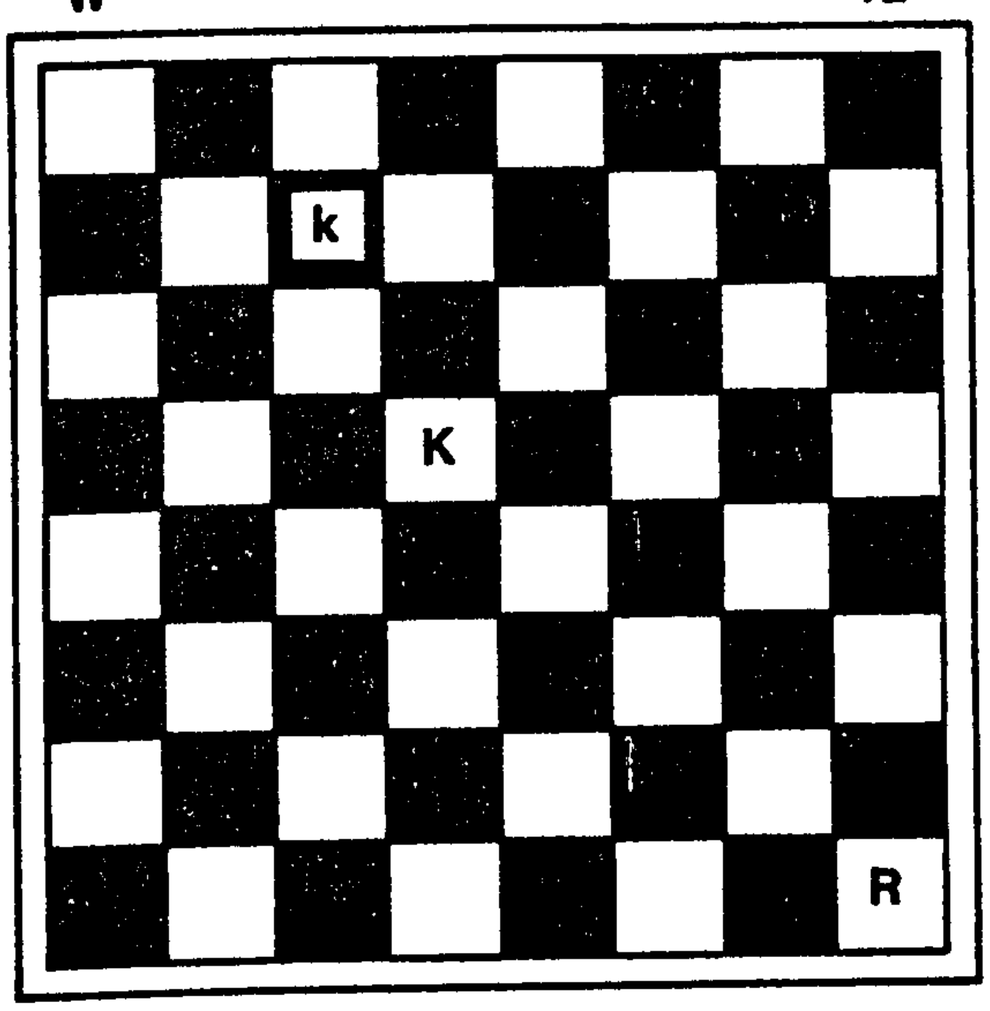
No associated functions are needed.

Although positions in class 12 can occur from other situations, the class will, in practice, be of interest almost exclusively in connection with the play from a position such as Figure 40.

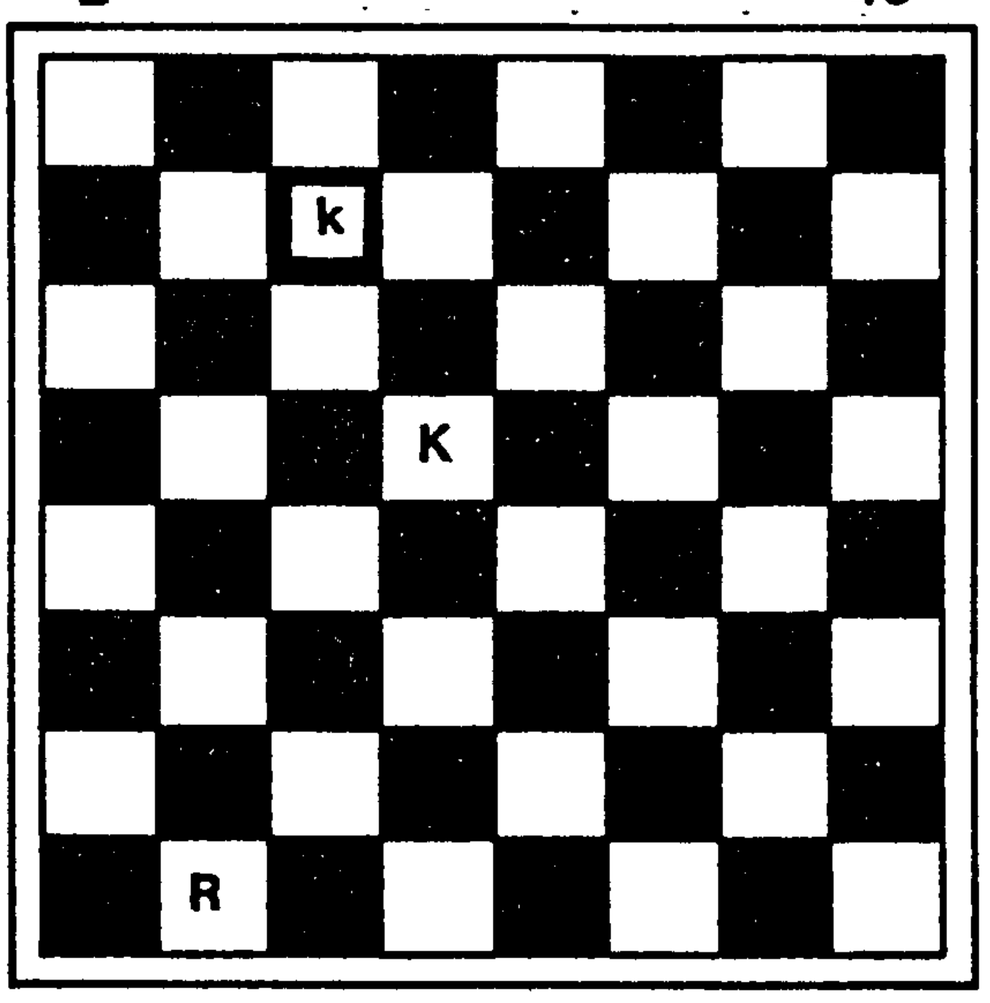
(5) An important pattern

Several cases were found in which the program's play could be considerably strengthened by recognising a further important configuration not explicitly discussed in Chess textbooks.

W 42



B 43



In Figure 42 the program, using the original algorithm, described in Section 4, will play the move 1. R-H6 (producing a position in class 7). Although this is a good move, a much stronger one is 1. R-B1, which gives Figure 43.

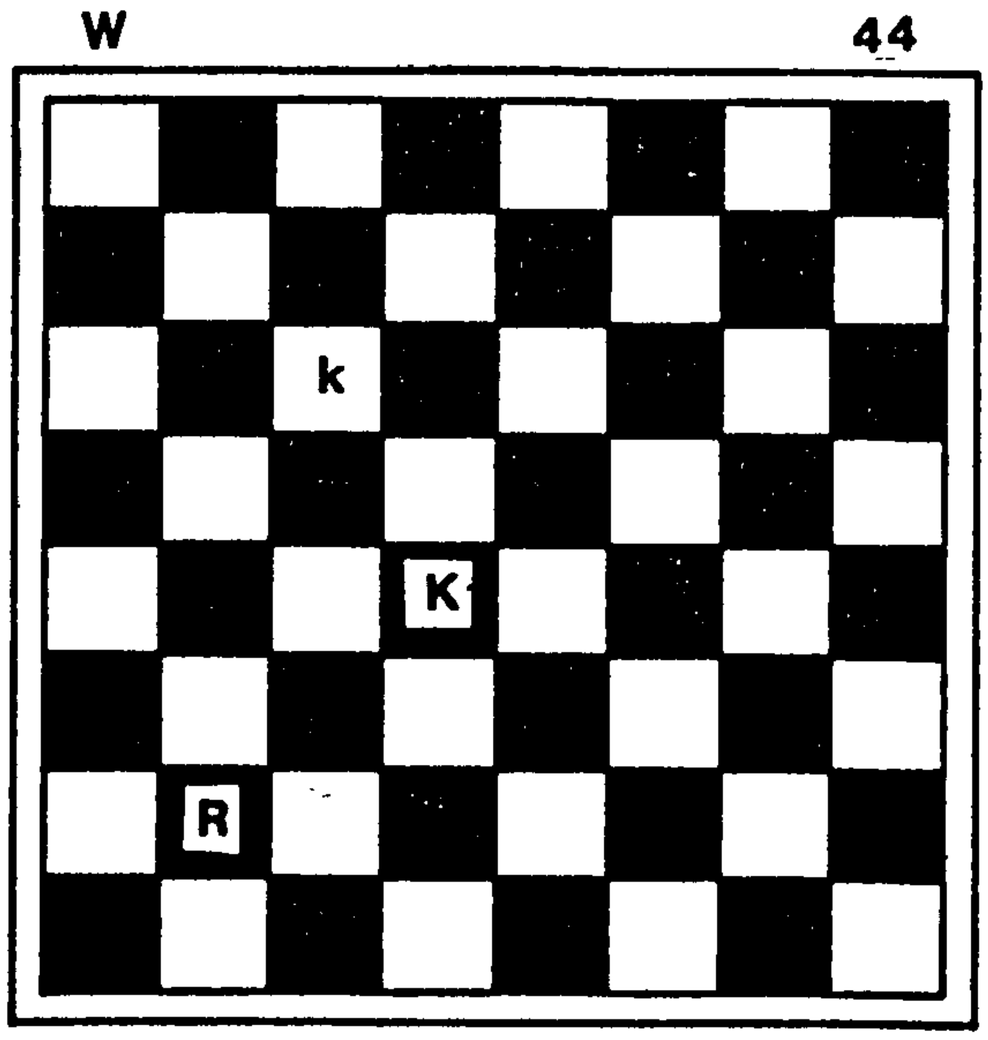
Black is now in considerable difficulty. If he plays K-C8, White replies 2. K-D6 (repeating the pattern) with mate in one to follow. If he plays instead K-D7, then White can force him back to the last rank by 2. R-B7 ch. Finally, if he plays K-D8, White can restrict him to one rank immediately by 2. R-B7. Similar situations occur frequently in practical play and to strengthen the program's play in these a

further equivalence class, class 13 was introduced, containing positions such as Figure 43 where the White King and the Rook are two files apart, with the Black King on the file between them, two ranks above the White King (and positions equivalent to these by symmetry). An inspection of the play from Figure 42 shows that class 13 needs to be ranked above class 7 and, in fact, because the pattern represented is so important the class has been ranked between classes 12 and 5.

With the orientation used for class 4, the definition of the corresponding rule is:

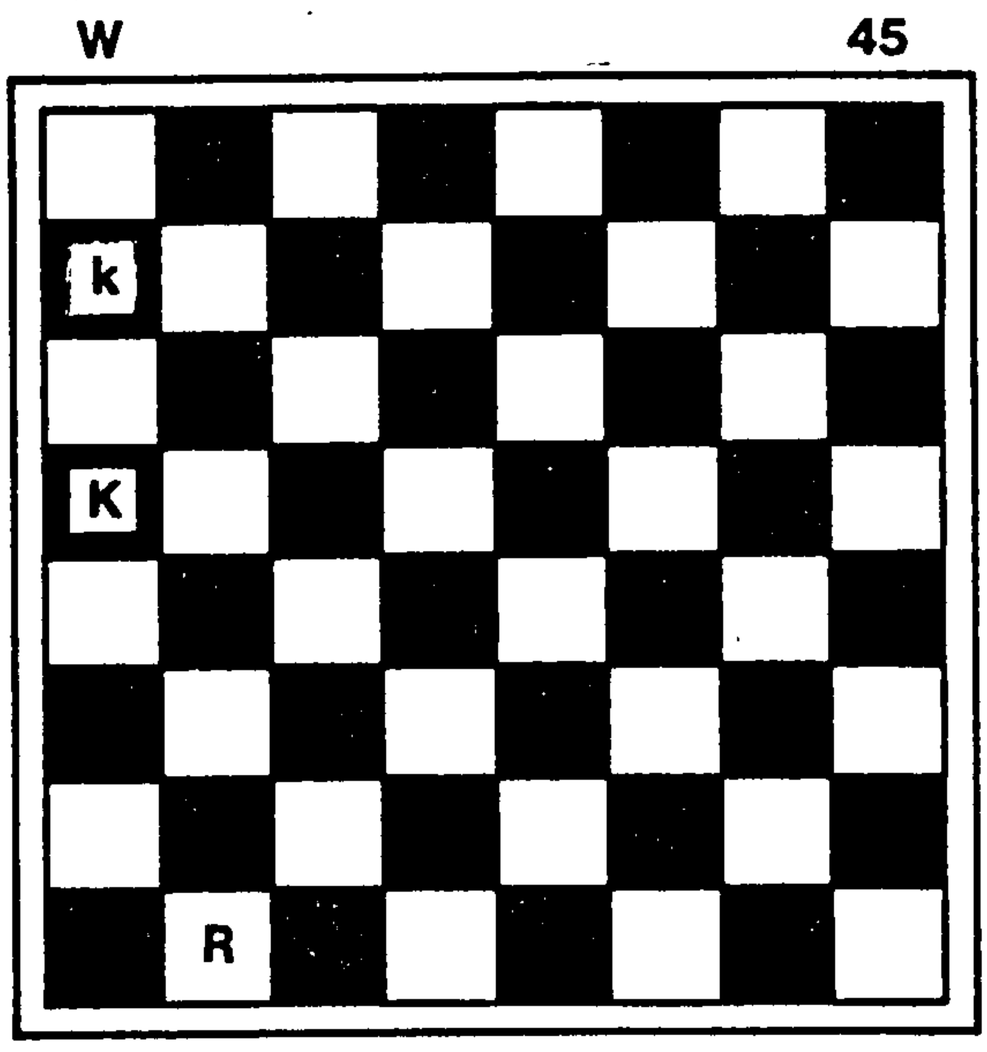
WK1=BK1+1 AND BK2=WK2+2 AND BK1=WR1+1

There is one associated function, with value 8-WR2 (priority is to be given to the position with the smallest value of WR2). Thus, in Figure 44, say, the program will 'lose a tempo' by 1. R-B1, that is, it will keep the Rook as close to the bottom edge of the board as possible. The alternative 1. R-B8 would be less precise, as Black could then force the Rook to move again by K-C7.



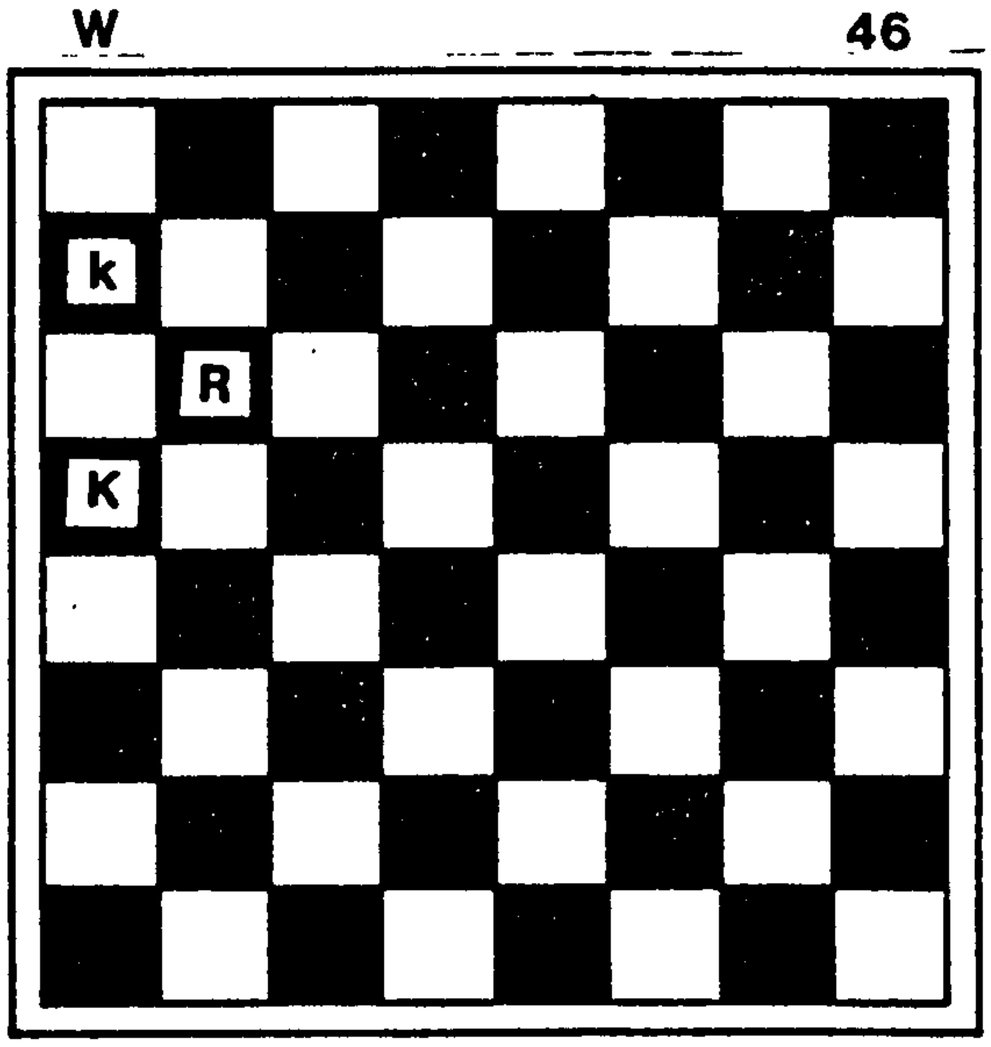
(6) Black King on the edge of the board

A number of difficulties arose in connection with the Black King restricted to the edge of the board by the Rook, which seemed to require small adjustments to the definitions of several of the classes. Unfortunately, these adjustments would have led to further problems in turn, and hence a new equivalence class was introduced to avoid the problems consequent on changing the definitions of existing classes. With the new class included in the algorithm all the difficult positions were handled satisfactorily.



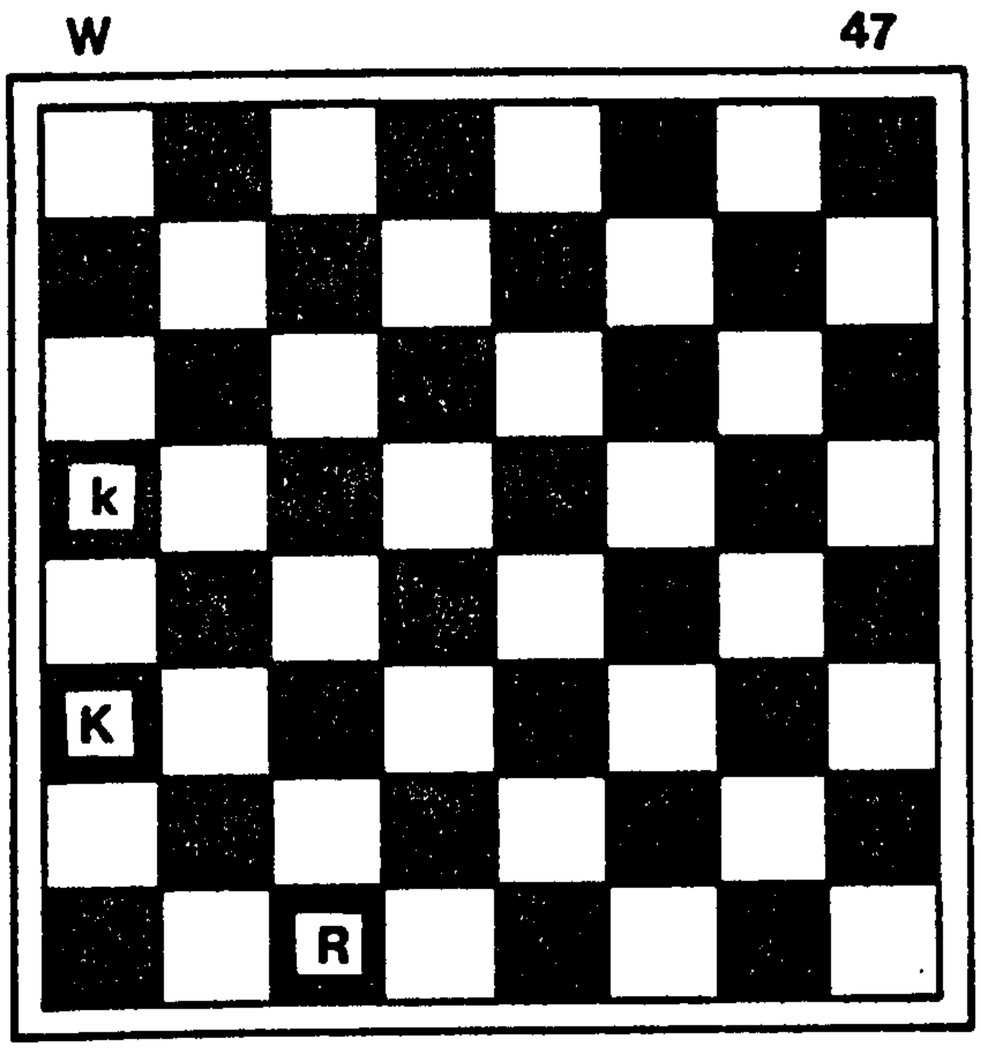
In Figure 45, White's best play is to lose a 'tempo' by 1. R-B2 (or B3,B4,B5). Then after Black's forced reply K-A8 White wins by K-B6 as in Figure 40. White should not play 1. R-B6 from Figure 45, since then after Black plays K-A8, White can now no longer reply K-B6. The B6 square is occupied by his own Rook. After 2. R-B1 (or B2,B3,B4,B5) and Black's forced reply K-A7, a further move 3. R-B6 will result in a repetition of position, with an eventual draw to follow. It would appear that it is necessary actively to prevent the move R-B6 in a position such as Figure 45. Since the positions arising after R-B2,B3,B4 or B5 from Figure 45 are

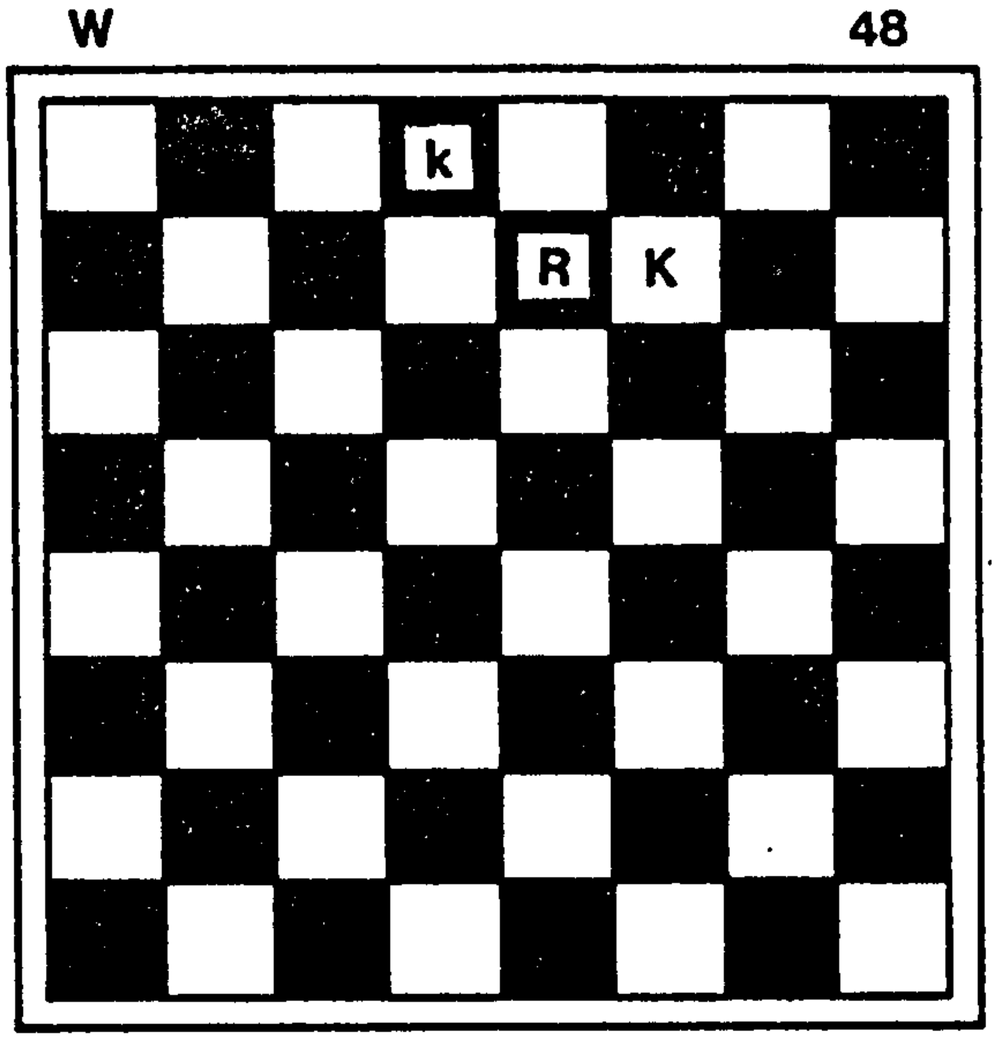
members of class 10, it is necessary to ensure that the position after R-B6 is not a member of any higher - ranked class. This requires a change to the definition of Rule 6 to rule out the case with both Kings on the Rook file. With this change the position would become a member not of class 6 but of class 8 and so the definition of rule 8 would need to be changed also and, similarly, the definition of rule 9.



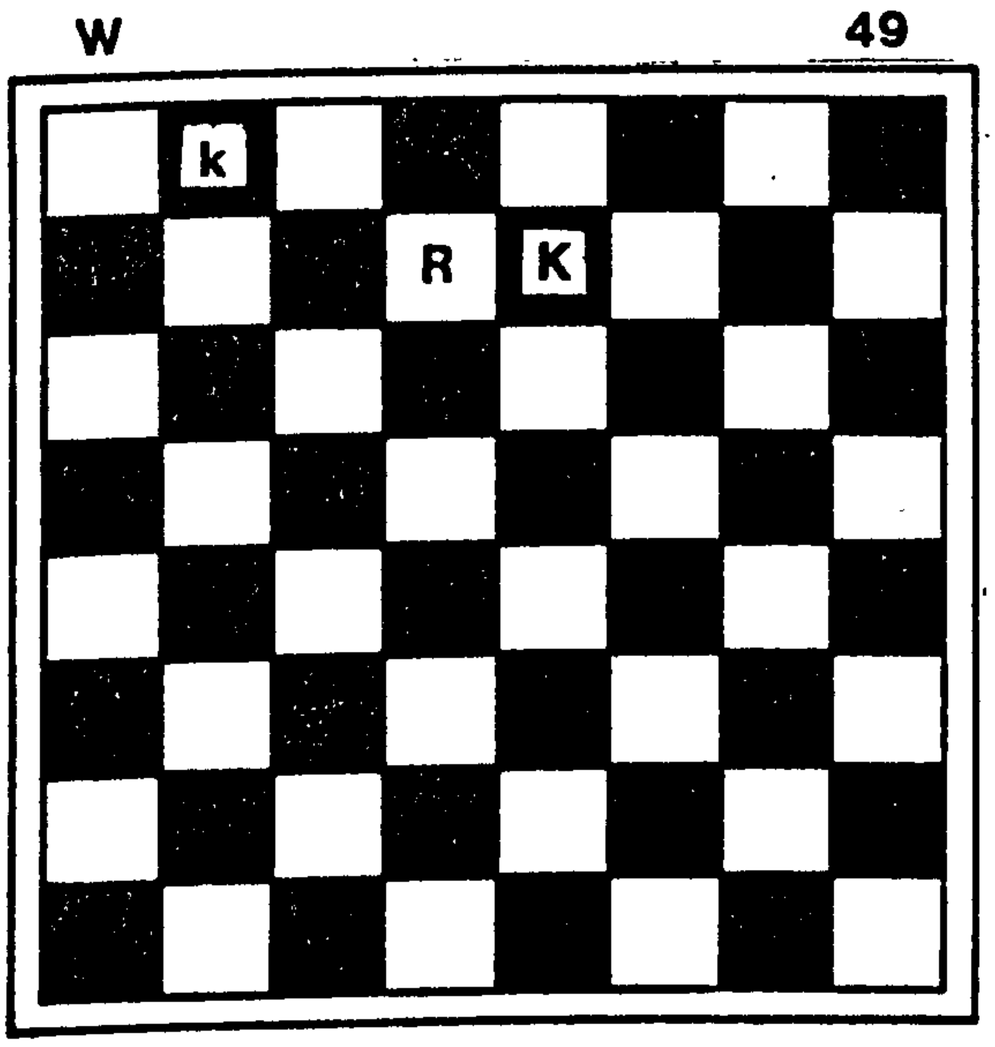
Unfortunately, however, there are further complications. In the critical position Figure 46, White should retreat his Rook along the B-file, giving a position in class 10, as already mentioned, not play K-B5. Consequently the position arising after 1. K-B5 must be eliminated from the higher ranked class 7. Changing the definition of this class requires a further change to the definition of class 9, to eliminate the position from that class also.

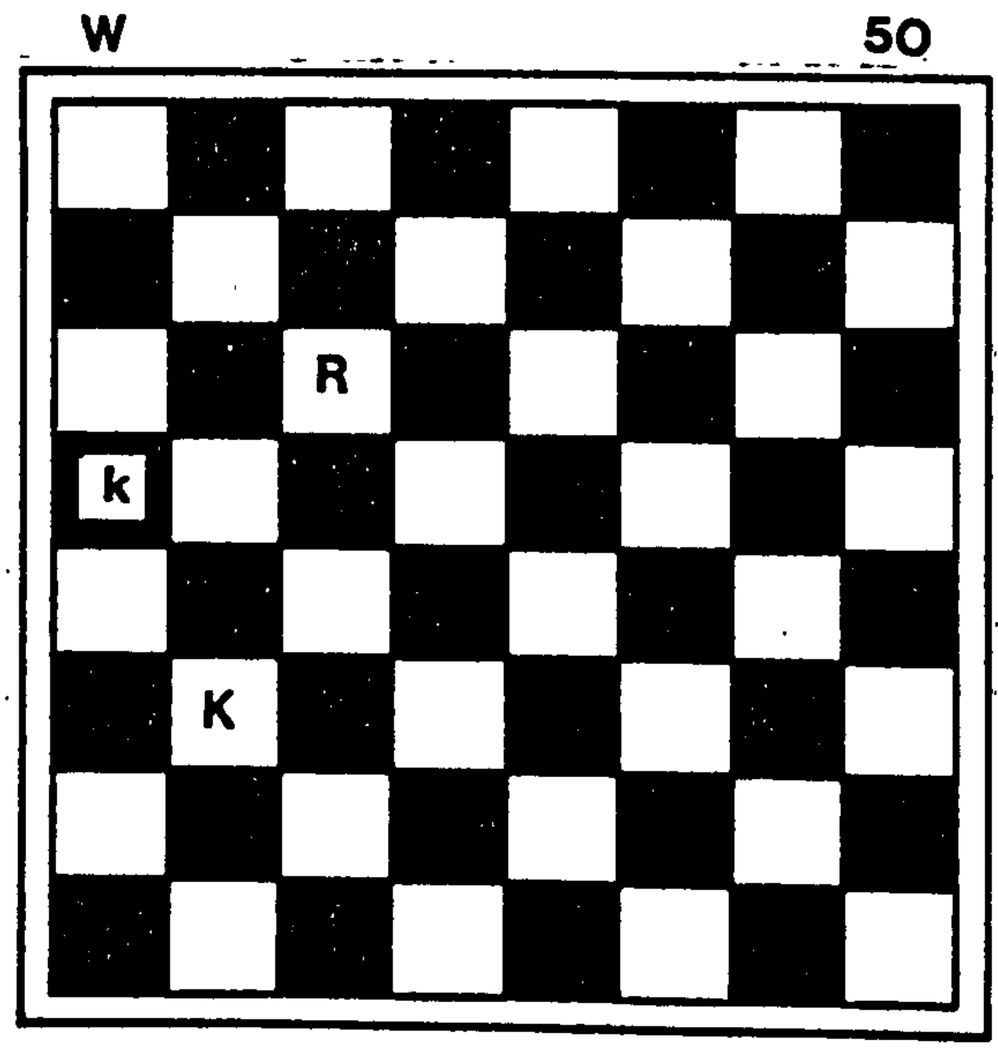
In fact, however, even these changes are not sufficient for precise play in all situations. Four more of the possible problems which can occur are described below.





In Figure 47, White's most direct winning move is 1. R-B1, not 1. R-C5 ch. Thus, it would seem that positions with both Kings on the Rook file must be eliminated from class 5. Moreover, in Figure 48, 1. K-F8 is better than 1. K-E6, so that positions with the Black King on the eighth rank would also need to be eliminated from class 7. (The position arising after K-E6 would otherwise be a member of class 7.)





In Figure 49, White should play 1. K-D8, not K-D6 so positions with the Black King on the eighth rank would also have to be eliminated from class 9.

Consider however Figure 50. Here it would be desirable to include the position arising after R-C5 ch in class 5 (since R-B6 simply loses the Rook), but this position would seem better excluded in the case where the Rook begins on C4 not C6 (since R-B4 is then superior to R-C5 ch).

From the above discussion, it should be clear that there are a large number of difficult situations which can occur with the Black King on the Rook file (or the first or eighth rank) and a correspondingly large number of cases in which the algorithm, as previously given, will produce a poor or imprecise move.

It is this kind of situation which it is important to detect when evaluating the effectiveness of a particular algorithm. If program correction and improvement entails a large number of piecemeal alterations, then the algorithm is probably suspect. Since the general performance of the algorithm under discussion is good, it should be possible to make the amendments required to handle the few difficult positions in an integrated and relatively simple manner.

If this proves to be impossible, then the appropriateness of the underlying equivalence class model may itself be called into question.

In the case under discussion, all the difficulties described can be overcome simply by making none of the modifications suggested and introducing instead a single new equivalence class, class 14. The class is defined (with the same orientation as class 4) by the predicate.

WK1=BK1 AND WK1=1 AND BK2=WK2+2 AND WR1=2.

The class is ranked between classes 13 and 5 and has one associated function, with value 8 - WR2 (that is, the Rook should be as far down the file as possible). With this one change to the program, a close inspection will show that all the difficult situations discussed above will be satisfactorily handled. Moreover the class has the advantage of being easily interpreted in terms familiar to chess players: "the Kings on the Rook file, two squares apart, with the Rook on the file next to them." Once again, however, such positions are not explicitly mentioned in chess textbooks.

6.2 Summary of changes made and the revised algorithm

The changes made to the original algorithm, as described in Section 4, are summarised below for convenience, followed by the revised algorithm in tabular form.

(a) New Classes: 12,13,14, ranked (in that order) between classes 4 and 5. With the orientation used for class 4, these are defined as follows.

Class 12

BK1=1 AND BK2=8 AND WK1=2 AND WK2=6 AND WR1=2. (No associated functions.)

Class 13

WK1=BK1+1 AND BK2=WK2+2 AND BK1=WR1+1,(One associated function, with value 8-WR2.)

Class 14

WK1=BK1 AND BK2=WK2+2 AND WK1=1 AND WR1=2.

(One associated function, with value 8-WR2.)

(b) Changes to existing classes

Note that only additions to existing rules are given. In each case the full rule is the logical AND of the previously given rule and the addition stated below.

Class 5	BK2≥5
Class 6	BK2≥5
Class 7	BK2≥5
Class 8	BK2≥5
Class 9	BK2≥5 AND WR1≤5
Class 10	A fourth associated function has been added, with value dist (Black King, Rook).

Table 5 Equivalence classes for King and Rook against King (revised algorithm)

Class	Property of position q (Black to move)	Class Value	Associated Functions f_1	f_2	f_3	f_4
1	White's Rook is en prise (i.e. may be immediately captured).	2	0	0	0	0
2	Black is checkmated.	14	0	0	0	0
3	Black is stalemated.	1	0	0	0	0
4	Black cannot avoid checkmate in one move.	13	0	0	0	0
12	The Black King is in the corner of the board, the White King and Rook are on the Knight file with the King on the sixth rank.	12	0	0	0	0
13	The Black King is on the file between the White King and the Rook and two ranks above the White King.	11	7	0	0	0
14	The Kings are in opposition on the Rook file, with the Rook on the adjacent Knight file.	10	7	0	0	0
5	The Kings are in vertical opposition (i.e. they are two squares apart on the same file) and Black is in check along the rank. Black is in the top half of the board.	9	0	0	0	0
6	The Kings are in vertical opposition with the Rook on the rank between them, one file closer to the centre. Black is in the top half of the Board.	8	0	0	0	0
7	The Kings are two ranks apart with the Rook on the rank between them; the White King is one file closer to the centre than the Black King and the Rook is not precisely one file from the White King. Black is in the top half of the board.	7	2	0	0	0

continued overleaf

Table 5 (continued)

Class	Property of position q (Black to move)	Class Value	Associated Functions f_1	f_2	f_3	f_4
8	The Kings are in vertical opposition with the Rook one file closer to the centre. Black is in the top half of the board.	6	0	0	0	0
9	The Kings are two ranks apart, with the Rook on the rank between them; the Rook controls a "good" quadrant. Black is in the top half of the board and the Rook is on the Queen's side or centre files.	5	2	3	0	0
10	The Black King is confined to a quadrant of the board and cannot immediately escape.	4	1	4	5	6
11	Always true.	3	0	0	0	0

Table 6 Associated functions for King and Rook against King (revised algorithm)

Function	Value of Function
1	8 - min ((WR1-1), (8-WR2))
2	50 - squad where squad = (8-WR1) × (8-WR2) } WR1 < BK1 (WR1-1) × (8-WR2) } WR1 > BK1
3	8 - abs (WK1-BK1)
4	8 - max {abs(WK1-BK1), abs(WK2-BK2)}
5	8 - min {abs(WK1-BK1), abs(WK2-BK2)}
6	max {abs(BK1-WR1), abs(BK2-WR2)}
7	8-WR2

6.3 The basic primitives

Rules

Each of the predicate functions defining the 14 rules given in Table 5 can be broken down into a number of simpler predicates, 'primitives', joined by the logical operators AND, OR and NOT.

These primitives all take the form

term relational operator value

where term is one of the following

WK1, BK1, WR1, WK2, BK2, WR2, WK1 - BK1, WR1 - WK1, BK1 - WR1,

BK2 - WK2, WR2 - WK2, BK2 - WR2, abs (WK1 - WR1), dist (Black King, Rook),

dist (White King, Rook) and dist (Black King, Rook) - dist (White King, Rook);

relational operator is =, ≠, >, <, ≥ or ≤

and value is an integer from 0 to 8 inclusive.

Thus the definition of rule 8 can be written as

$$(BK2 \geq 5)\ \text{AND}\ (WK1 - BK1 = 0)\ \text{AND}\ (WR1 - WK1 = 1)$$

From the above list it will be seen that term is either one of the six possible co-ordinates of the three pieces, the difference between any pair of such values (or its modulus), the block distance between two pieces or the difference between two such distances. Although some possible combinations, such as abs (WK1 - BK1), have not been used the total number of possible combinations is clearly fairly small. In a self-modifying system of the kind discussed in

subsequent sections it would be possible for all the possible terms to be available to the system in advance.

Associated functions

Each associated function is of the form

expression

or constant - expression

depending on whether the largest or the smallest value of expression is to be taken.

expression is in general either

(i) a term of the kind described above (e.g. WR2 or dist (Black King, Rook)).

(ii) the distance between a piece and a specified edge (e.g. the distance between the Rook and the top of the board); or

(iii) the larger or smaller of two of the above.

Exceptionally, when an 'area' such as the number of squares in a quadrant is required, expression may be the product of two distances.

Thus, function 1 which is defined in Table 6 by

8 - min ((WR1 - 1), (8 - WR2))

is of the form constant - expression, where expression is the smaller of WR1 - 1 and 8 - WR2.

These are the distance of the Rook from the left-hand edge and the top edge of the board, respectively.

7. King and Rook against King : discussion of the changes made and the revised algorithm

In this section, the changes made to the original form of the algorithm and the significance of the testing carried out are discussed. The revised algorithm is then evaluated from a number of different viewpoints, in relation to Chess textbooks and some theoretical results of Clarke (1975). Some illustrative games played against human opponents are also given, followed by a discussion of Sections 5 - 7 in the light of the objectives set out in Section 1.

7.1 The changes made to the original algorithm

The 2486 positions in which moves were made by the original version of the program were stored on file, together with the program's move in each position. After the changes to the algorithm described above, the move played by the program in each of these positions was recalculated and compared with that originally made.

The outcome was that the changes made to the algorithm resulted in a different move being selected in 538 cases (22%).

In as far as this can be judged by inspection, in each of the 538 cases the new move is an improvement on the old, or one which appears equally good.

Although there may, of course, still be errors remaining in the algorithm, it is reasonable to say that the program's play has been significantly improved in a large proportion of cases by making just the few amendments given in Section 6.

It is also probably fair to say that the program now contains more

"knowledge" than is contained in a typical textbook description for this endgame.

Most of the examples of difficulties arising from the first version of the algorithm were situations which might equally well be mishandled by the average chessplayer with only the textbook information to guide him.

An inspection of the descriptions given in Chess textbooks, such as Fine (1941) and Golombek (1954), will show that no indication is given to the reader as to how to play in positions such as Figures 40, 42, 44, 45 or 46.

The original algorithm's difficulties in these positions led to the introduction of classes 12 - 14.

The ideas of "specifying a small quadrant", "restricting Black to the top half of the board" and "keeping the Rook at a distance", which led to changes in the definitions of classes 5 - 9 and an extra associated function being introduced for class 10, are not explicitly stated in the textbook descriptions, but are implicit in the examples given. For example, the Black King is always in such a position that any quadrant it is restricted to is a small one, etc.

7.2 The significance of the testing

The testing described in Section 6, although substantial, was both unsystematic and non-random: the human opponents generally chose their own initial positions and did not necessarily play the best moves for Black in every case.

It is therefore a posibility that some parts of the algorithm may have been tested inadequately or not at all. As a check on this, the following table of "selection levels" was produced to provide an indication of the degree to which each of the rules and functions in the algorithm has been tested.

Table 7 Selection levels for the revised algorithm

CLASS	Total No. of occurences	No. of times selected	Class value alone	SELECTED BY First Function	Second Function	Third Function	Fourth Function	TIES
1	7,874	0						
2	180	180	180					
3	202	0						
4	1,059	167	105					62
12	60	32	32					
13	199	125	105	20				
14	402	150	87	63				
5	70	67	67					
6	150	106	106					
7	1,444	258	169	76				13
8	336	55	55					
9	2,187	362	64	127	170			1
10	30,023	984	0	192	73	367	285	67
11	2,869	0						

Total 47,055 2486

2486 positions

The column headed "Total No. of occurrences" gives a breakdown of the class membership of each of the total of 47,055 successor positions (with Black to move) generated in the course of finding the "best" move for the 2486 initial positions (with White to move) presented to the program.

For each initial position investigated, one successor position was selected by the program as "best" and the class membership of these positions is broken down in the "No. of times selected" column.

The remaining six columns are derived from a comparison of the "best" and the "second best" successor position in each of the 2486 cases.

Thus, taking the example of class 9, of the 362 occasions when the best successor position belonged to class 9, on 64 occasions the second best position belonged to a lower-ranked class (so the associated functions had no effect on the selection).

In the remaining 298 (= 362 - 64) cases, the second best position was also a member of class 9, so one or more of the associated functions was needed to make a choice between them. In 127 cases the first function was sufficient to break the tie, but in 170 cases the second function was also needed. In one case both functions were insufficient and an "arbitrary" choice was made (a residual tie).

The thick line shows the number of functions associated with each class.

From an examination of the table, it can be seen that positions in every class have occurred as possibilities at some stage of the testing but that, as previously suggested, positions in classes 1 (Rook <u>en prise</u>),

3 (Stalemate) and 11 (residual positions) have never been chosen as best.

Nevertheless these three classes account for 10,945 successor positions (23% of the total) and it is useful to include them either so that they can be positively rejected by the algorithm (classes 1 and 3) or to ensure that all positions belong to some equivalence class (class 11).

Checkmate positions (class 2) are selected whenever they occur (180 cases). It is, in fact, impossible for more than one of the successors of any position with White to move to be a member of class 2.

Positions in the second highest-ranked class, class 4 (where Black cannot prevent mate in one) occur frequently (1,059 times) but appear to be relatively seldom selected as best (167 times). The explanation is that these positions often occur in comparison either with checkmate positions (which will always be chosen in preference to them) or with each other.

Positions in class 5 occur relatively seldom, but when they do they are usually important (selected 67 times out of 70 occurrences). Positions in class 10 are by far the most frequent to occur (64%) and to be selected (40%).

Together classes 1, 3, 10 and 11 account for a total of 40,968 successor positions (87%) and 984 of those positions selected (40%).

Although the other 10 classes represent only 13% of all successor

positions, they are selected as best in 60% of all cases. In many cases the membership of these classes is small, but with a high rate of selection (for example, positions in class 5 were selected on 96% of the occasions on which they occurred). Even class 4, which for reasons previously explained is selected relatively infrequently has a selection rate of 15% and the other 9 classes (2, 12 - 14 and 5 - 9) all have higher rates than this. By contrast, class 10 has a selection rate of only 3% and classes 1, 3 and 11 all have rates of zero. This suggests that classes 2, 4, 12 - 14 and 5 - 9 may represent positions which although relatively infrequent in occurrence are in some respects crucial to White's strategy.

The final five columns indicate the extent to which the choice of each of the associated functions for each class is reinforced by the testing carried out. Thus with a different choice for the fourth function associated with class 10, up to 352 (= 285 + 67) of the selected positions might not have been chosen (they are all cases where the best two positions were members of class 10 and the first three functions were insufficient to choose between them). With a different choice for the third function, up to 719 (= 367 + 285 + 67) of the selections might have been different, and so on. It would seem, therefore, that the choice of functions associated with class 10 is a very sensitive part of the overall algorithm.

The occurrence of 62 arbitrarily resolved ties between positions belonging to class 4 simply reflects the decision not to associate any functions with that class (since there is no practical advantage in choosing any one of the positions rather than another).

However the 67 ties for class 10 and, to a much lesser extent, the ties

for classes 7 and 9, may indicate that a further function or functions should be associated with these classes. In some cases it may be that the choice between tied positions was genuinely arbitrary; in others it may be that further discrimination was necessary for best possible play. Naturally the need for such discrimination may not be either apparent to the chessplayer or indicated by textbook descriptions in many cases.

Nevertheless, any further attempt to improve the algorithm might usefully begin by isolating these "tied" positions for further inspection.

7.3 Class membership of all positions with Black to move

For purposes of comparison, a table is given below showing the frequency of membership of each of the 14 equivalence classes in the revised algorithm, for a particular standard orientation of the board.

The orientation is taken from Clarke (1975) and can be obtained from any given position by a suitable combination of reflections about axes of symmetry.

With this orientation, the Black King is restricted to the triangle of squares A1, B1, C1, D1, B2, C2, D2, C3, D3, D4. The two White pieces can be anywhere on the board, except that when the Black King is on the diagonal A1, B2, C3, D4 the White King is reflected to lie on or below the diagonal A1 to H8, and when both Kings are on the diagonal the White Rook is reflected to lie on or below the diagonal.

This gives 21,959 legal positions with White to move, 189 of which are checkmates in one move, 587 are checkmates in two moves and 121 are checkmates in 16 moves, which is the maximum ever required.

White wins in all cases where it is his own move initially.

With Black to move there are 28,056 legal positions, in 27 of which he is already checkmated. There are 9 positions where Black is stalemated and 2787 where the Rook is en prise. In all other positions Black is lost. The above figures are those given by Clarke.

The following table shows the membership of each equivalence class over all positions with Black to move, using Clarke's orientation.

Table 8 Class membership for revised King and Rock against King algorithm

Class	Frequency
1	2787 (10%)
2	27
3	9
4	78
12	5
13	150 (1%)
14	30
5	81
6	11
7	78
8	57
9	145 (1%)
10	19735 (70%)
11	4863 (17%)

Total 28056 positions

Percentages of the total (28056), rounded to the nearest 1%, are given when 1% or greater.

It will be seen that the frequency of membership of class 1 (Rook en prise), class 2 (Black is checkmated) and class 3 (Black is stalemated) agrees with Clarke's figures.

The figures in the table would seem generally to support the conclusions reached in Section 7.2, i.e. that class 5 etc. "represent positions which although relatively infrequent in occurrence are in some respects crucial to White's strategy".

However, it should be noted that the tables are not strictly comparable since each successor position (with Black to move) can arise from many different initial positions. Hence the same position may be included more than once in Table 7.

Clarke's orientation also introduces a bias into the percentage frequencies, since not all of the 28,056 positions represent 8 positions in the full set of positions with Black to move, although most do. It is unlikely, however, that this bias is of more than minor importance.

7.4 The Choice of Algorithm

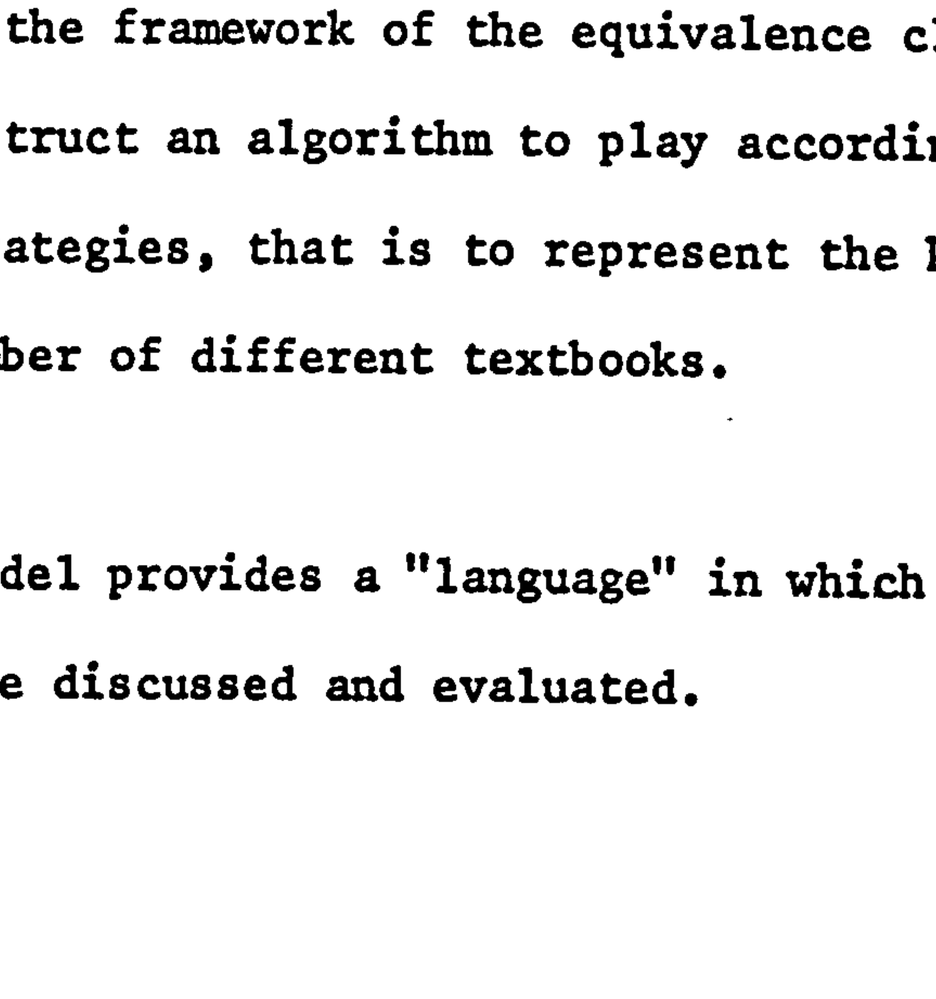
The algorithm developed in Section 4 and revised in Section 6 corresponds to a composite description of White's strategy derived from several textbooks and refined in the light of experience.

In this section two examples where the play given in particular textbooks differs from that of the revised algorithm are briefly discussed in terms of the overall equivalence class model and the changes to the algorithm which would be necessary to produce the textbook moves.

If the descriptions given in different textbooks are compared, it will be seen that there are often differences in the details of the advice given by one author and another. It might reasonably be expected that in certain positions the move played would vary from one author to another, that is that each author uses a slightly different algorithm, some of which will be more precise than others.

However, within the framework of the equivalence class model, it is possible to construct an algorithm to play according to a variety of different strategies, that is to represent the knowledge given in any one of a number of different textbooks.

Moreover, the model provides a "language" in which alternative strategies can be discussed and evaluated.



As an example, consider Figure 51. Fine (1941) gives the following play from this position (in modified algebraic notation) :

1. K - B2, K - D4; 2. K - C2, K - E4; 3. K - C3, K - E5;

4. K - C4, K - E4; 5. R - E1ch; K - F5; 6. K - D4, K - F4;

7. R - F1ch, K - G5; 8. K - E4, K - G6; 9. K - E5, K - G5;

10.R - G1ch, K - H4; 11.K - F5, K - H3; 12.K - F4, K - H2;

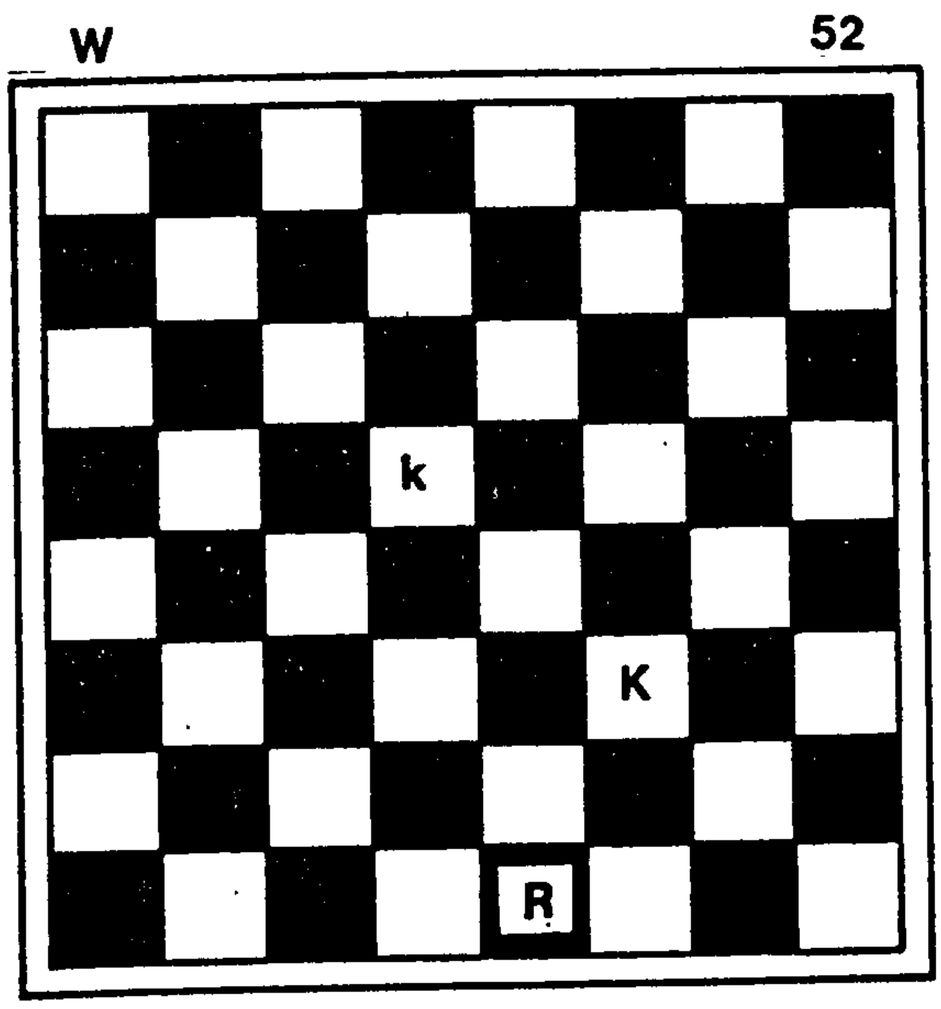
13.R - G3, K - H1; 14.K - F3, K - A2; 15.K - F2, K - H1;

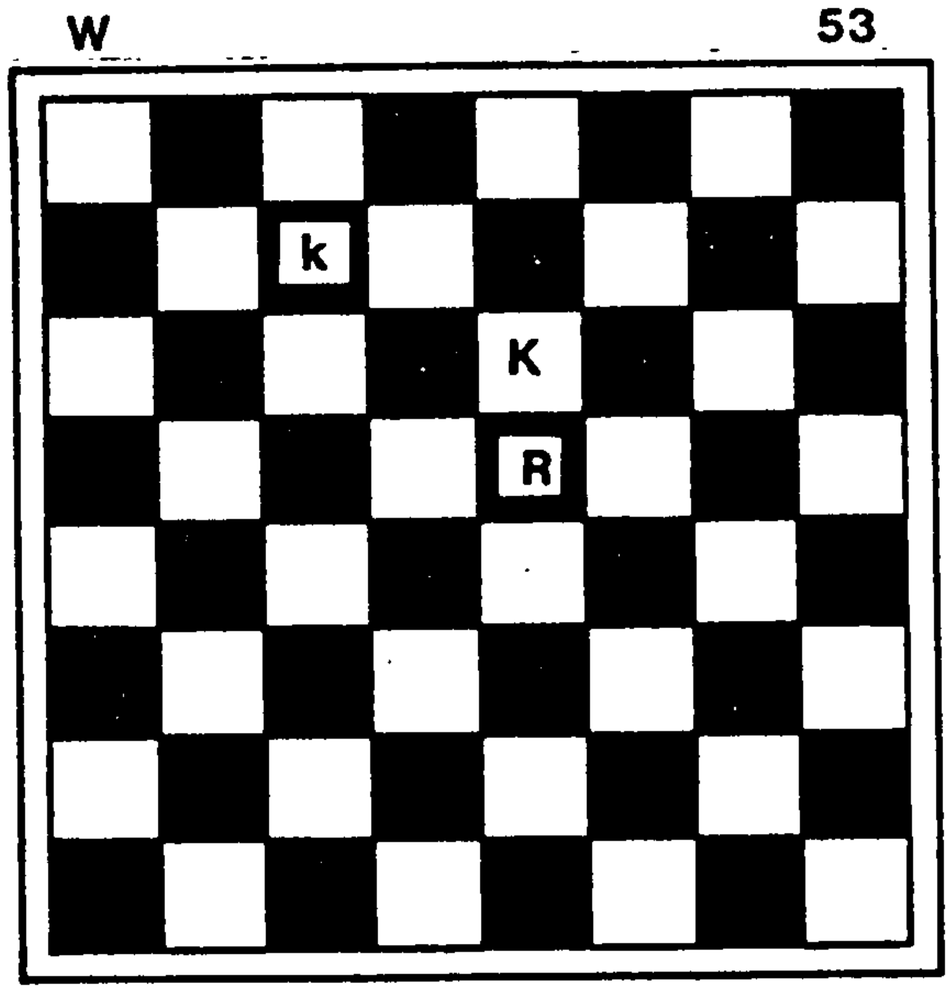
16.R - H3 Mate.

The final version of the algorithm as described in Section 6 finds the same moves 5 - 16 as those given by Fine but its moves in each of the first four positions with White to move would be different. The algorithm would, in fact, play R - E1, R - E1, R - D1, and R - D1 respectively. (The program's second move is actually better than the book move, requiring one move less to checkmate, according to Clarke (1975).) In each case, both the algorithm's move and Fine's move produce positions which are members of class 10.

In the terms of the algorithm, Fine is applying the following criteria to decide between positions in class 10: bring the Kings as close together as possible and, subject to that, restrict the Black King with the Rook.

One simple way of modifying the algorithm to achieve this effect would be to change the first three associated functions from 1, 4, 5 (in that order) to 4, 5, 1.





As a second example, consider Figure 52.

In this position Golombek (1954) gives the move R - E4 (R - K4 in descriptive notation): "cutting the Black King off from White's half of the board". The algorithm, however, evaluates the resulting position as belonging to class 9 and selects K - F4 in preference, giving a position belonging to class 7. One way of modifying the algorithm to ensure that R - E4 is selected is simply to delete class 7 altogether. With this modification the algorithm plays the book moves in every position of the main variation quoted by Golombek, with the exception of Figure 53 (White's eighth move). Here Golombek gives 8. R - C5ch (8. R - B5ch in descriptive notation), whereas the algorithm assigns the resulting position to the class of "residual positions", class 11, and chooses 8. R - D5 (giving a position in class 9).

To modify the algorithm to play 8. R - C5ch it would be necessary to define a new equivalence class, of which the position after that move had been played was a representative member, ranking higher than class 9.

Whether the moves given by Golombek are better or worse than the program's moves is irrelevant here. The series of moves correspond to two different strategies, either of which could be represented within the overall framework of the model.

In general, many (possibly all) textbook strategies can be implemented using the same model and any of these could then serve as the basis for further experimentation and subsequent refinement.

7.5 Automatic program verification

Clarke (1975) gives a table of the number of positions in which White needs 1, 2, 3, ..., 16 moves to checkmate from the 21959 positions with White to move using his standard orientation. Although, as has previously been pointed out, not all of Clarke's positions correspond to the same number of positions in the full set of positions with White to move, proportions calculated from this table are nevertheless likely to be a good estimate of the true proportions for all positions.

As a further experiment in program verification, a system was set up to play a series of games against each of two set Black strategies A and B, described below. Two hundred "automatic" games were played against each Black strategy, taking (legal) starting positions generated at random from the full set of positions with White to move. The games were played out to completion (with a maximum of 25 moves permitted for each side).

In each case, the game ended in checkmate in sixteen moves or less (as mentioned previously, Clarke has shown that sixteen is the largest

number of moves ever required by a perfect strategy).

The two Black strategies, which were implemented using the equivalence class model, were as follows:

(A) take the Rook if possible, otherwise move towards the centre of the board, and

(B) take the Rook if possible, otherwise move at random.

The second part of strategy A required the maximization of the following two associated functions in turn:

(i) $$8 - \underline{\max}\left\{\frac{(5 - WR1)\ (4 - WR1)}{2}, \frac{(5 - WR2)\ (4 - WR2)}{2}\right\}$$

and (ii) $$8 - \underline{\min}\left\{\frac{(5 - WR1)\ (4 - WR1)}{2}, \frac{(5 - WR2)\ (4 - WR2)}{2}\right\}.$$

The left-hand side of Table 9 shows the frequency (as a percentage) of games in which White checkmated in 1, 2, 3, ..., 16 moves with each of the two strategies and the approximate theoretical frequencies, calculated from Clarke's figures (to two places of decimals). The right-hand side of the table shows the cumulative frequencies (checkmate in n moves or less), which are likely to be of more significance since they are less affected by random fluctuations.

It must be emphasized that each game played has been recorded only once. Thus a game in which White checkmated in seven moves, say, has been included as a checkmate in <u>seven</u> moves only, although there were, of course, necessarily intermediate positions in which White won in 6, 5, 4, 3, 2 and 1 moves. These latter position do not, however, form a random sample from the set of all positions with White to move.

Table 9 Checkmate in n moves: strategies A and B and (approximate) theoretical value, C

n	Frequencies (%)			Cumulative Frequencies (%)		
	A	B	C	A	B	C
1	0.5	0.0	0.86	0.5	0.0	0.86
2	2.5	3.0	2.67	3.0	3.0	3.53
3	2.0	2.5	2.20	5.0	5.5	5.73
4	3.5	3.0	1.08	8.5	8.5	6.81
5	6.5	5.0	2.76	15.0	13.5	9.57
6	12.0	9.0	4.97	27.0	22.5	14.54
7	16.0	17.5	6.46	43.0	40.0	21.0
8	10.5	14.5	9.79	53.5	54.5	30.79
9	8.5	9.5	11.45	62.0	64.0	42.24
10	6.5	12.5	10.85	68.5	76.5	53.09
11	9.5	10.0	11.68	78.0	86.5	64.77
12	6.5	8.0	12.25	84.5	94.5	77.02
13	10.0	3.5	10.17	94.5	98.0	87.19
14	3.5	1.0	9.23	98.0	99.0	96.42
15	1.0	0.5	3.01	99.0	99.5	99.43
16	1.0	0.5	0.55	100.0	100.0	99.98

A - Black Strategy A (move towards the centre)

B - Black Strategy B (random)

C - (approximate) theoretical values, calculated from Clarke (1975)

An inspection of this table suggests that the final form of the algorithm plays very well indeed. For example, 43% of games are won in seven moves or less against strategy A and 40% against strategy B, compared with 21% for the theoretical value. 94.5% of games against strategy A and 98% against strategy B are won in 13 moves or less, compared with a theoretical value of only 87.19%.

Doubtless these figures partly reflect the fact that Black's play is not optimal, although strategy A would seem to be a strategy offering good practical chances in play.

Four hundred games would also seem to be ample to detect any instances of infinite repetitions of position. However, none of these occurred. Indeed no case was found where more than the theoretical maximum of sixteen moves was needed to checkmate, although of course it is not certain that the positions in which sixteen moves were taken are those in which that number of moves is required by theory.

It would be interesting to obtain comparable figures for other King and Rook against King algorithms.

However, it has not been possible to find any such information in the literature.

7.6 Illustrative games

Some games played by the revised form of the program against human opponents are given below, to illustrate some of the most commonly occuring situations. The program finds moves for White (the side with the Rook) only, with Black moves supplied by a human opponent. A complete endgame is played from any legal starting position, with White to move, chosen by the opponent. Alternatively the opponent

may choose a random (legal) starting position if he wishes.

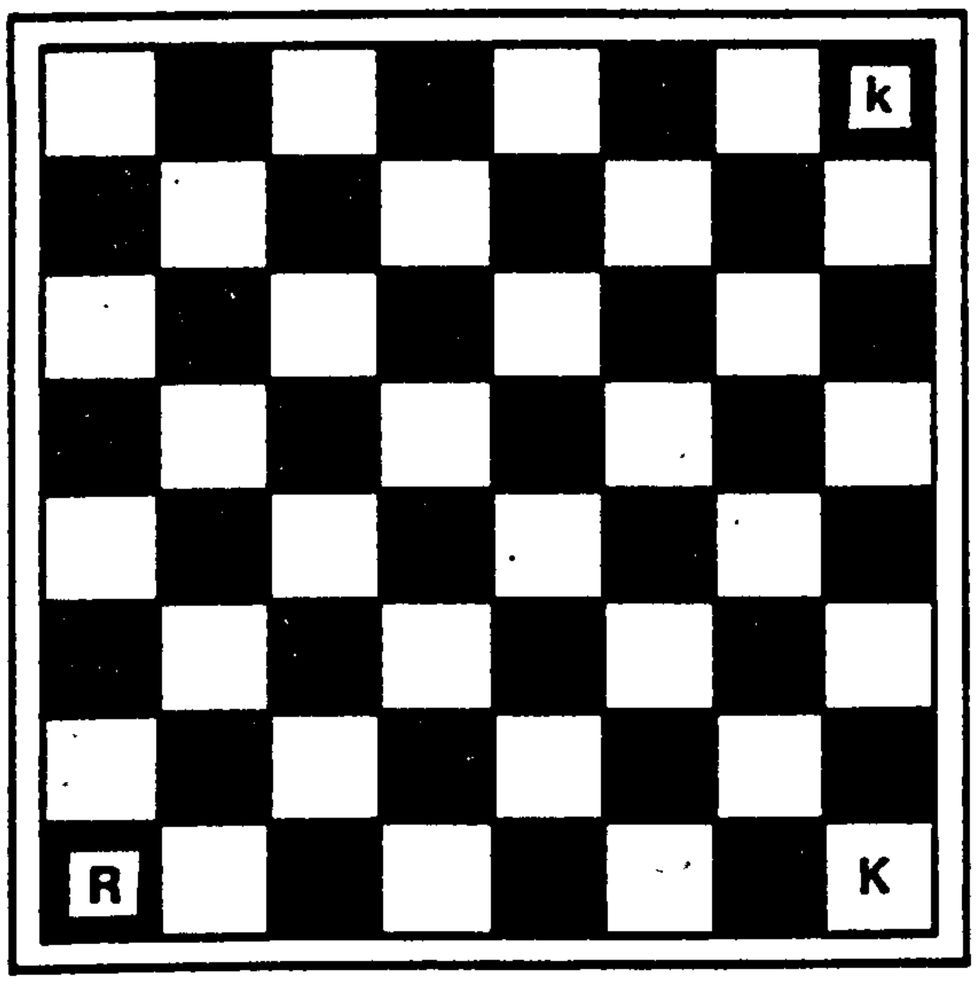
The 50 - move drawing rule and the "three-fold repetition of position" drawing rule have not been implemented, as they present no problems of theoretical interest in this context.

No case has so far been found where the program takes more than the theoretical maximum necessary number of 16 moves before checkmate or where a particularly poor move has been made.

Games are given both in the modified form of algebraic notation used in the preceding sections and in the descriptive notation, which is the one most commonly used in English speaking countries.

The figure in parentheses after each pair of moves is the equivalence class of the position resulting after White's move.

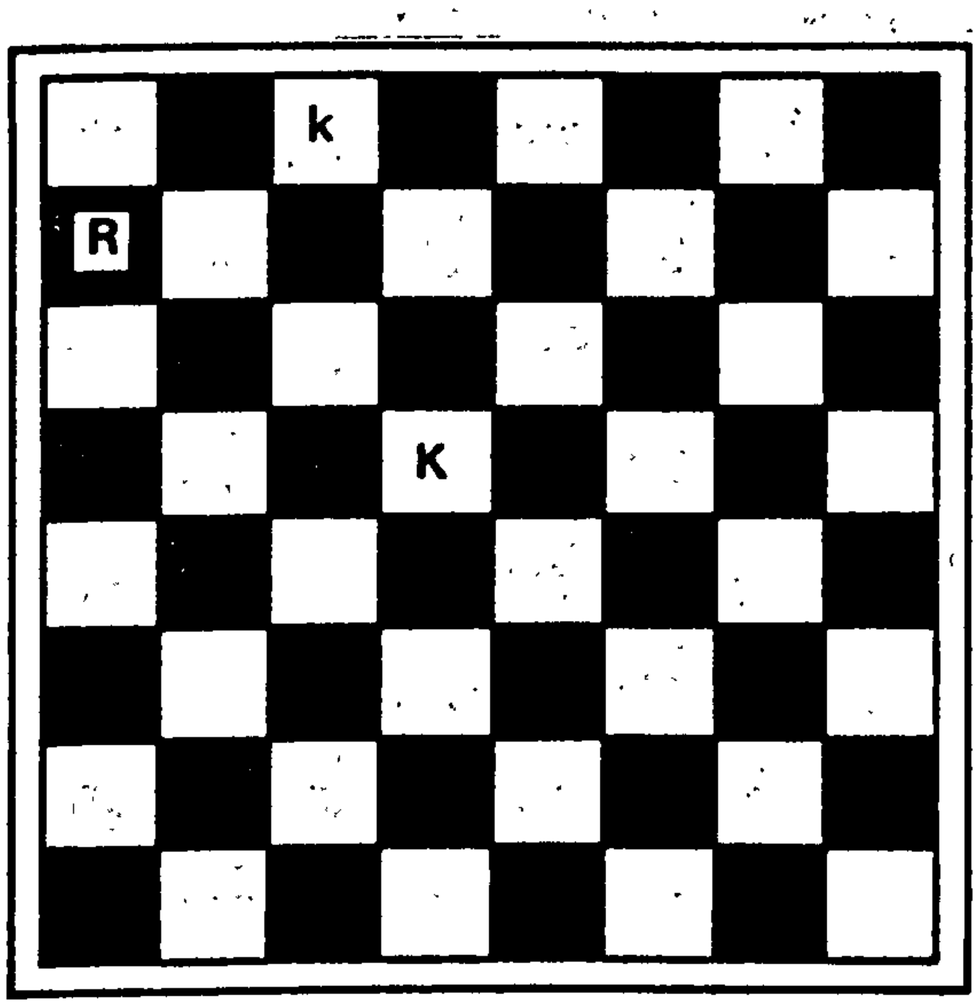
Game 1



1.R-A7	K-G8	1.R-QR7	K-N1	(10)
2.K-G2	K-F8	2.K-N2	K-B1	(10)
3.K-F3	K-E8	3.K-B3	K-K1	(10)
4.K-E4	K-D8	4.K-K4	K-Q1	(10)
5.K-D5	K-E8	5.K-Q5	K-K1	(10)
6.K-D6	K-F8	6.K-Q6	K-B1	(7)
7.K-E6	K-G8	7.K-K6	K-N1	(7)
8.K-F6	K-H8	8.K-B6	K-R1	(7)
9.K-G6	K-G8	9.K-N6	K-N1	(4)
10.R-A8	Mate	10.R-R8	Mate	(2)

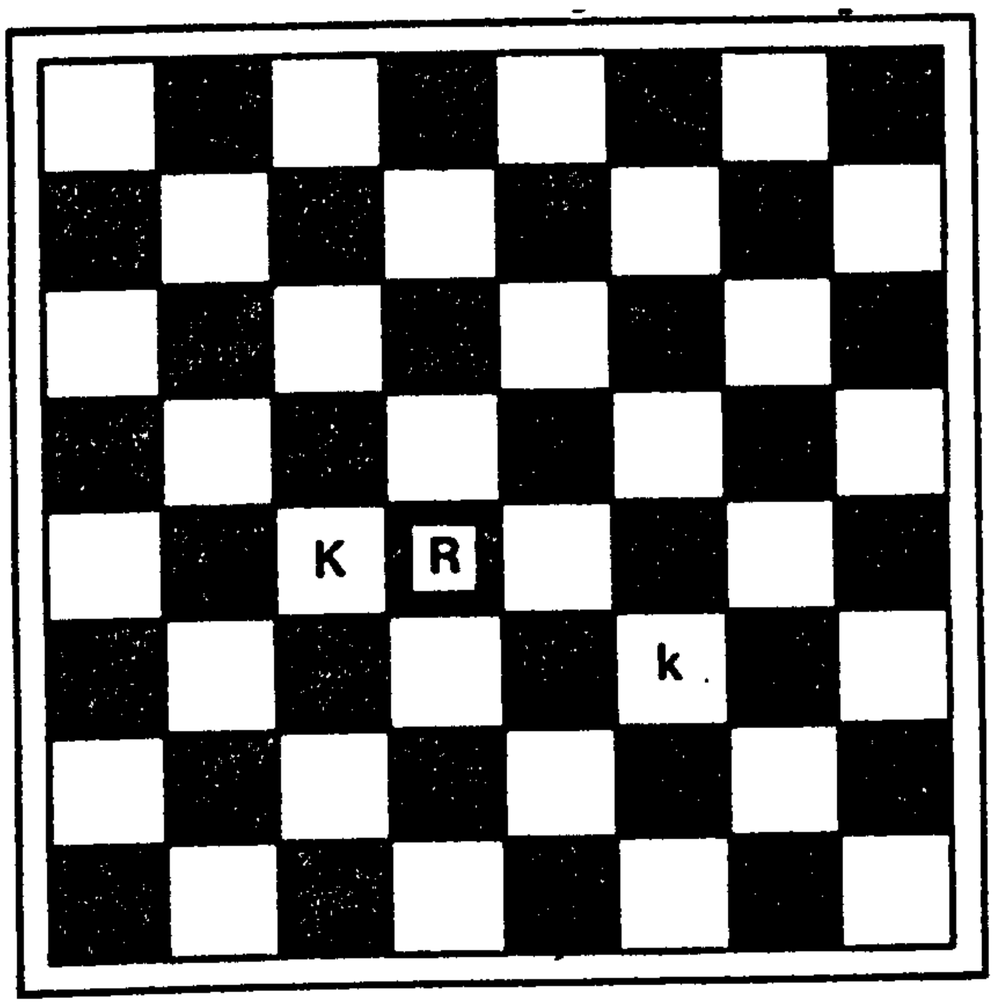
Game 2

Starting from the previous position, Black deviates at move 5, playing 5... K-C8 (5....K-B1, in descriptive notation). The game continues from the diagrammed position



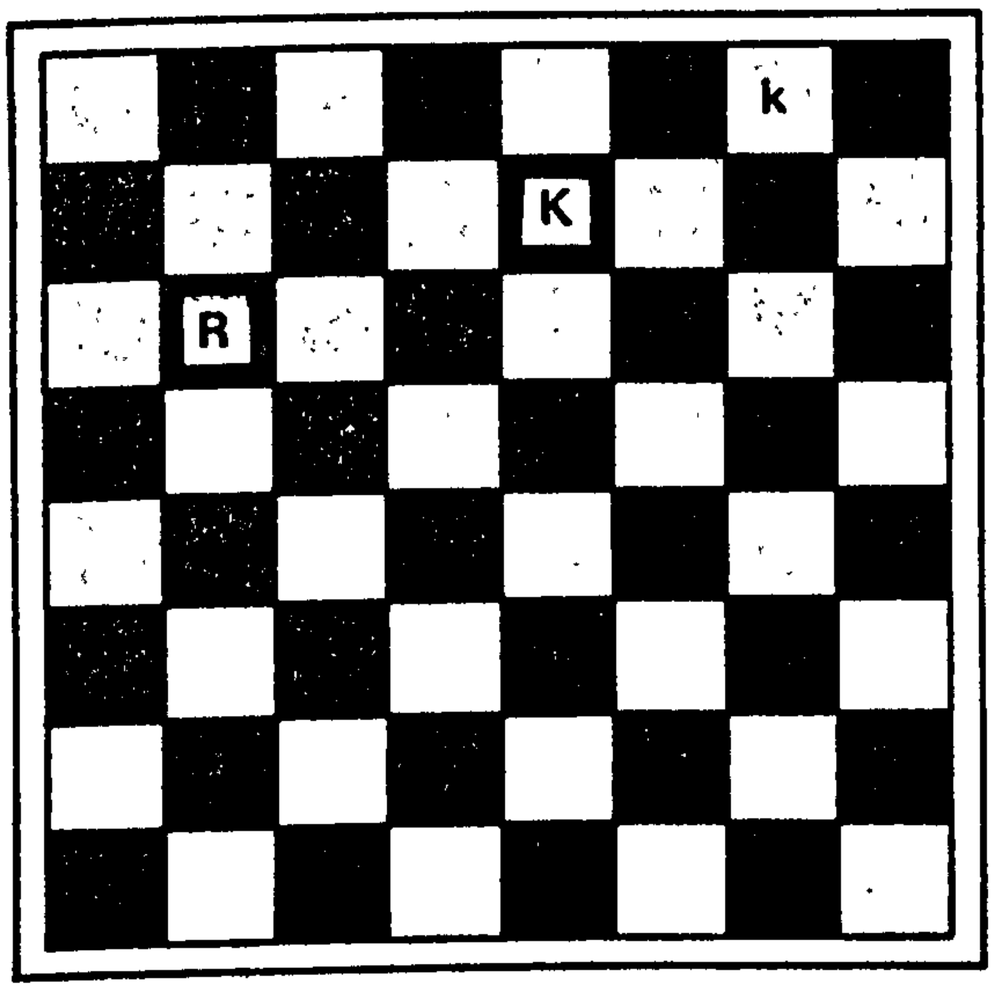
6.K-D6	K-B8	6.K-Q6	K-N1	(7)
7.R-C7	K-A8	7.R-QB7	K-R1	(9)
8.K-C6	K-B8	8.K-B6	K-N1	(9)
9.K-B6	K-A8	9.K-N6	K-R1	(4)
10.R-C8	Mate	10.R-B8	Mate	(2)

Game 3



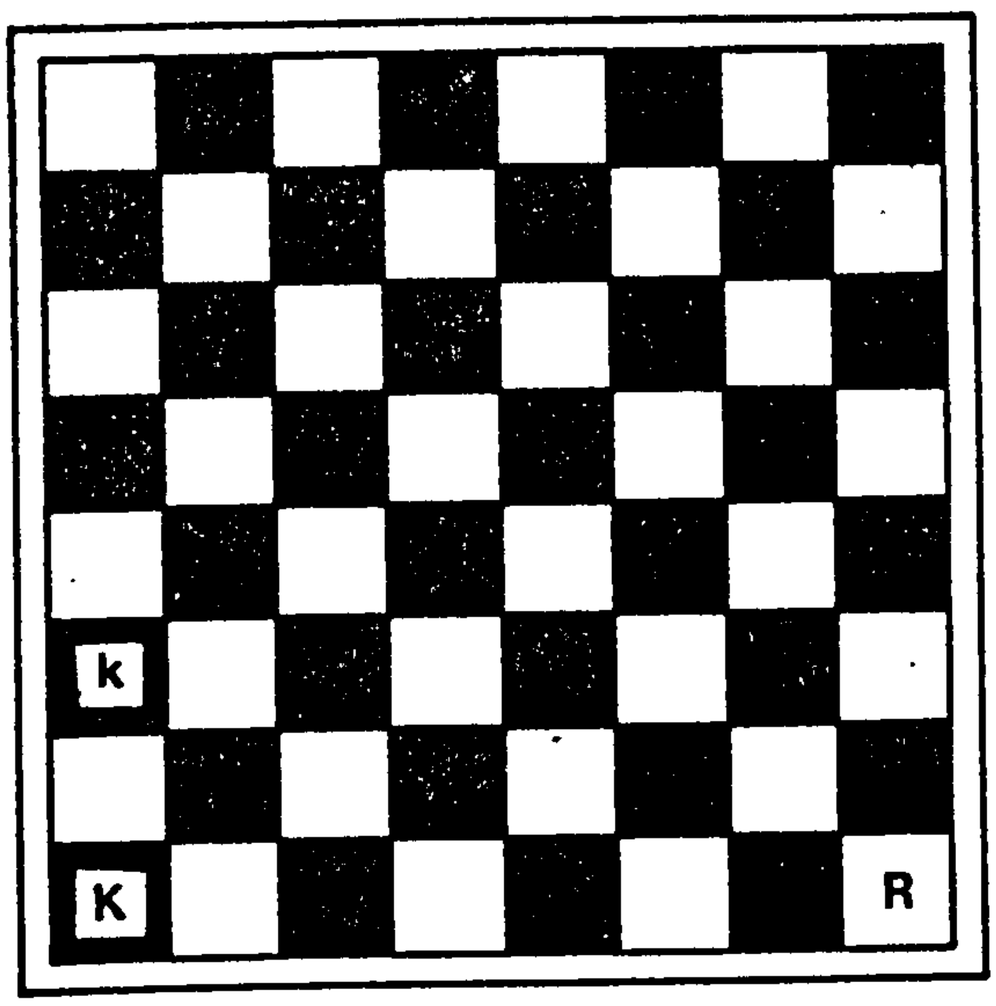
1.K-D3 K-F2	1.K-Q3 K-B7	(8)
2.R-E4 K-F3	2.R-K4 K-B6	(9)
3.K-D4 K-F2	3.K-Q4 K-B7	(7)
4.R-E3 K-G2	4.R-K3 K-N7	(9)
5.K-E4 K-F2	5.K-K4 K-B7	(9)
6.K-F4 K-G2	6.K-B4 K-N7	(6)
7.R-F3 K-H2	7.R-B3 K-R7	(7)
8.R-G3 K-H1	8.R-N3 K-R8	(9)
9.K-F3 K-H2	9.K-B3 K-R7	(9)
10.K-F2 K-H1	10.K-B2 K-R8	(4)
11.R-H3 Mate	11.R-R3 Mate	(2)

Game 4



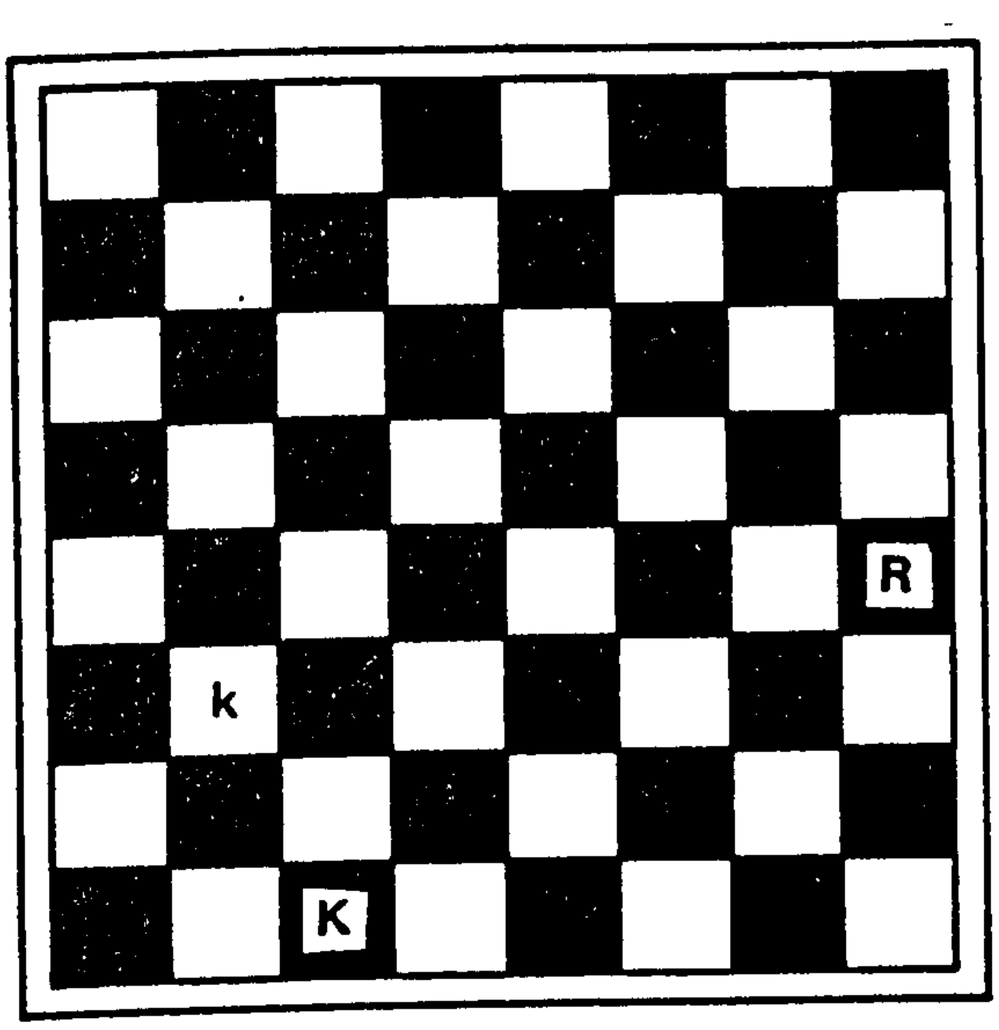
1.R-G6ch K-H7	1.R-KN6ch K-KR2	(5)
2.K-F7 K-H8	2.K-KB7 K-KR1	(4)
3.R-H6 Mate	3.R-KR6 Mate	(2)

Game 5



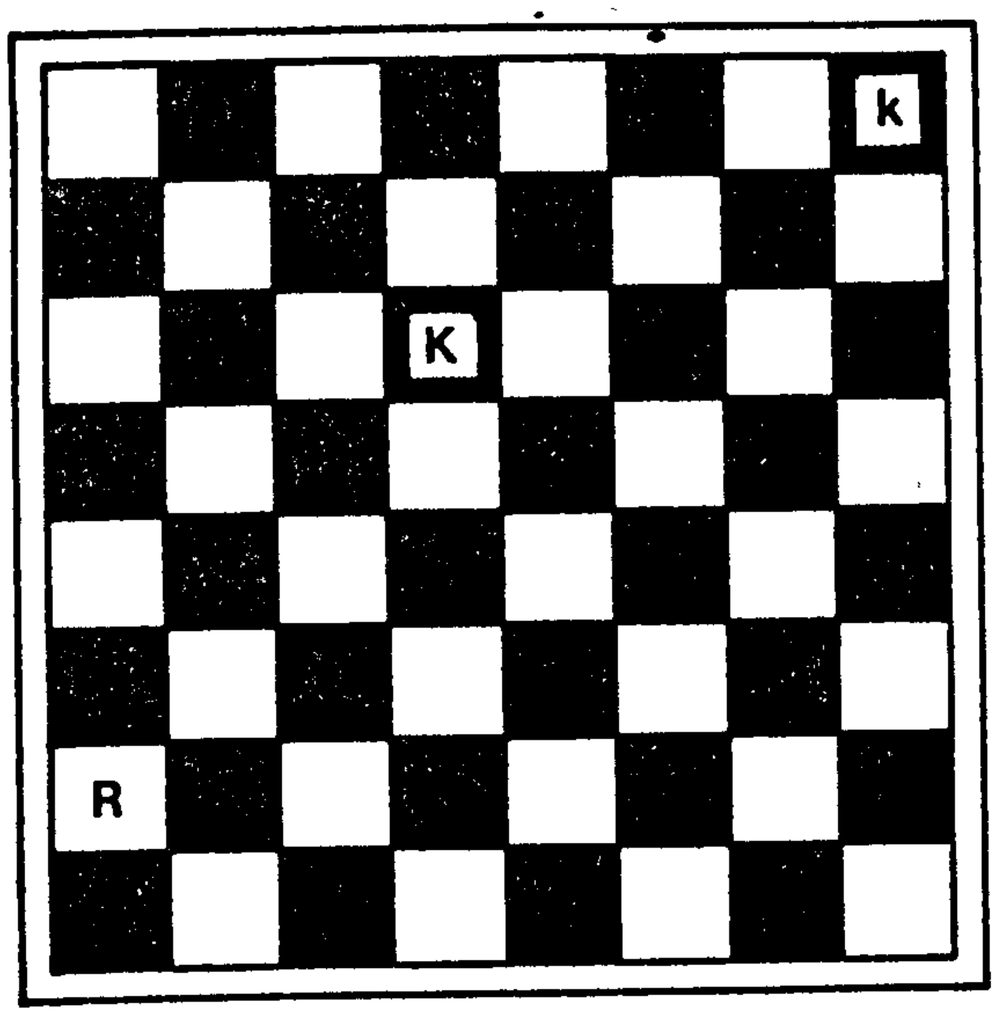
1.R-B1 K-A4	1.R-QN1 K-R5	(14)
2.K-A2 K-A5	2.K-R2 K-R4	(14)
3.K-A3 K-A6	3.K-R3 K-R3	(14)
4.K-A4 K-A7	4.K-R4 K-R2	(14)
5.K-A5 K-A8	5.K-R5 K-R1	(14)
6.K-B6 K-B8	6.K-N6 K-N8	(12)
7.R-C1 K-A8	7.R-QB1 K-R8	(4)
8.R-C8 Mate	8.R-B8 Mate	(2)

Game 6



1.K-D2 K-B2	1.K-Q2 K-N7	(13)
2.R-B4ch K-A3	2.R-N4ch K-R6	(5)
3.K-C3 K-A2	3.K-B3 K-R7	(6)
4.R-B3 K-A1	4.R-N3 K-R8	(7)
5.K-C2 K-A2	5.K-B2 K-R7	(9)
6.R-C3 K-A1	6.R-QB3 K-R8	(4)
7.R-A3 Mate	7.R-R3 Mate	(2)

Game 7



1.R-A7 K-G8	1.R-QR7 K-KN1	(10)
2.R-E7 K-F8	2.R-K7 K-KB1	(9)
3.K-D7 K-G8	3.K-Q7 K-KN1	(7)
4.K-E8 K-H8	4.K-K8 K-KR1	(14)
5.K-F7 K-H7	5.K-KB7 K-KR2	(12)
6.R-E6 K-H8	6.R-K6 K-KR1	(4)
7.R-H6 Mate	7.R-KR6 Mate	(2)

7.7 Discussion

This section continues the discussion of Section 4.6 in light of the material contained in Sections 5 - 7.

The objective of the testing described in these sections was to investigate how readily the algorithm given in Section 4 could be improved in response to weaknesses in its play identified during testing against human opponents.

One of the criteria stated in Section 1 was that an algorithm:

"should be capable of relatively straightforward modification ... and the changes required should be indicated by the deficiencies observed in [its] performance".

The analysis in Section 6 would seem to establish that this criterion is satisfied in this case and gives a practical demonstration of the means by which such modifications can be performed.

The changes made are entirely within the framework of the model described in Section 2, that is they consist simply of adding new equivalence classes and associated functions and modifying the definitions of some of the existing classes.

The end result is still a "compact" algorithm with only 14 equivalence classes, and the evidence in Sections 6 and 7 suggests that its performance is now extremely good and superior in some positions to many textbooks. The complexity of the algorithm is still apparently commensurate with that of textbook descriptions and no tree-searching is used.

A further aim stated in Section 1 was "to produce quantifiable results wherever possible" and a number of tables and figures have accordingly been presented to enable the revised algorithm to be compared with others in any subsequent work in this area and with the theoretical results of Clarke(1975).

The rules defining the equivalence classes and the associated functions have been shown to be composed of elementary primitives in simple combinations. In the case of King and Rook against King it would not seem difficult to specify a list of all possible primitives which might occur and thus provide a basis for subsequent computer generation of combinations of primitives to form the rules defining equivalence classes.

The question of program improvement forms the major theme of the remainder of this thesis, where an algorithm for a more complex endgame, King and Pawn against King, is discussed. This endgame is used partly to demonstrate the applicability of the equivalence class model to other endgames and partly because the difficulties which occur in refining the algorithm - in this case to play <u>perfectly</u> in a given set of positions - are more severe than for King and Rook against King. The discussion is presented from the viewpoint of the possible future development of a fully automatic system to refine an initial algorithm within the framework of the equivalence class model, and some of the requirements for such a system are discussed.

8. An Algorithm for the Endgame King and Pawn against King

In the first part of this thesis an algorithm for the endgame King and Rook against King was discussed, firstly as a means of demonstrating the conversion of the playing rules given in chess textbooks into an algorithm, and secondly to investigate how easily errors and weaknesses in the algorithm found as the result of testing against human opponents could be rectified. Both these objectives can be considered to reflect on the question of the appropriateness or naturalness of the 'equivalence class' model described in Section 2 as a basis for producing algorithms for Chess endgames. The second part of the thesis is devoted to an analysis of an algorithm for a further endgame, King and Pawn against King, partly as a second illustration of the use of the model but principally to investigate the problems and techniques involved in modifying an algorithm of this kind to play perfectly. Some of the problems inherent in such a task are discussed in Section 5. Although in the experimentation described in Section 10, modifications were made simply by changing the program coding or entries in tables etc. "by hand", it is hoped that the results obtained will provide a starting point for subsequent research aimed at developing programs in this area which can improve their own performance substantially on the basis of their past experience (or other knowledge supplied to them, such as "illustrative games" etc.)

In Section 11 the results obtained are summarised from the viewpoint of self-improving programs. The experimentation to be carried out is specified in Section 9 and the objectives are discussed further there. In this section an initial form of the King and Pawn against King algorithm is given, with some explanation and justification of the choices made. Essentially, however, this form of the algorithm is

to be considered as arising at an intermediate stage of development, that is there are some grounds for considering it is a reasonable algorithm, but it is still subject to errors or omissions. As will be seen in subsequent sections, there are a number of important deficiencies in the algorithm as given.

There are fifteen equivalence classes defined for this version of the algorithm, with a total of seven different associated functions. As for the King and Rook against King algorithm, these are based on an analysis of the discussion and examples given in a number of textbooks for this ending.

An investigation of these descriptions shows that the difficult positions for this endgame arise when the Black King blocks the Pawn's advance. White's strategy then is to take the opposition and thus to force Black to either retreat or move to one side. In the latter case, the White King moves forward to the other side to support the Pawn's advance. The equivalence classes given represent a detailed working out of this general strategy, together with other more obvious objectives such as "promote the Pawn if you can do so safely" and "do not leave the Pawn <u>en prise</u>", etc.

For simplicity, it is assumed that White wins (and the game is over) if he advances his pawn to the eighth rank and it is not then immediately capturable by the Black King. (The small number of instances in which the Pawn promoting to a Queen would give stalemate can be safely ignored; White can always choose to promote to a Rook instead.)

8.1 The equivalence classes

The rules defining the fifteen equivalence classes are summarised in the table below, in the order in which they are to be evaluated, followed by an explanation of some of the terms used in the table and further discussion of some of the principal classes.

In implementing the rules (and those defined subsequently in this thesis) it has been assumed that the Pawn is in one of the left-most four files of the board. Any position can be transformed into this form by at most one reflection about the (vertical) centre line of the board.

The notation used for defining the rules and associated functions is the same as for King and Rook against King except that WR1 and WR2 do not, of course, occur and the Pawn's file and rank are denoted by WP1 and WP2, respectively.

Table 10 King and Pawn against King (initial algorithm)

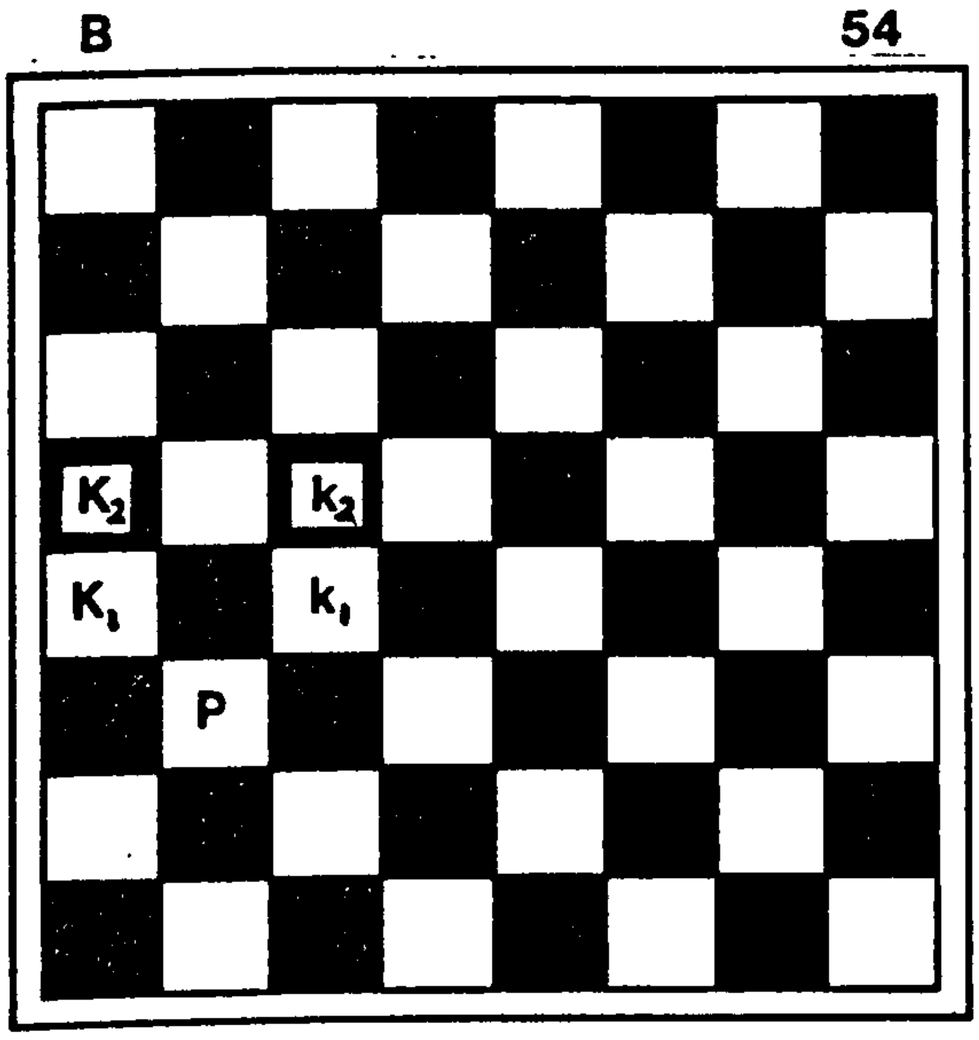
Class	Property of position 2 (Black to move)	Class value
1	The pawn is en prise (i.e. can be immediately captured)	1
2	Black is stalemated	2
3	The pawn is on the eighth rank	15
4	The pawn can 'run'	14
5	The Black King is closer to the pawn than the White King	3
6	The Black King can occupy the square in front of the Pawn	4
7	The White King is on the square in front of the Pawn and Black can take the opposition	5
8	The White King is two or more files closer to the Pawn than the Black King and not below the rank of the Pawn	13
9	The position satisfies pattern A	12
10	The position satisfies pattern B	11
11	The White King is somewhere in front of the pawn, on the same file	10
12	The Kings are in opposition and the White King is on a critical square, but not on the Pawn's file.	9
13	The Kings are in opposition and the White King is not on a lower rank than the Pawn	8
14	The White King is on a critical square.	7
15	(Always true)	6

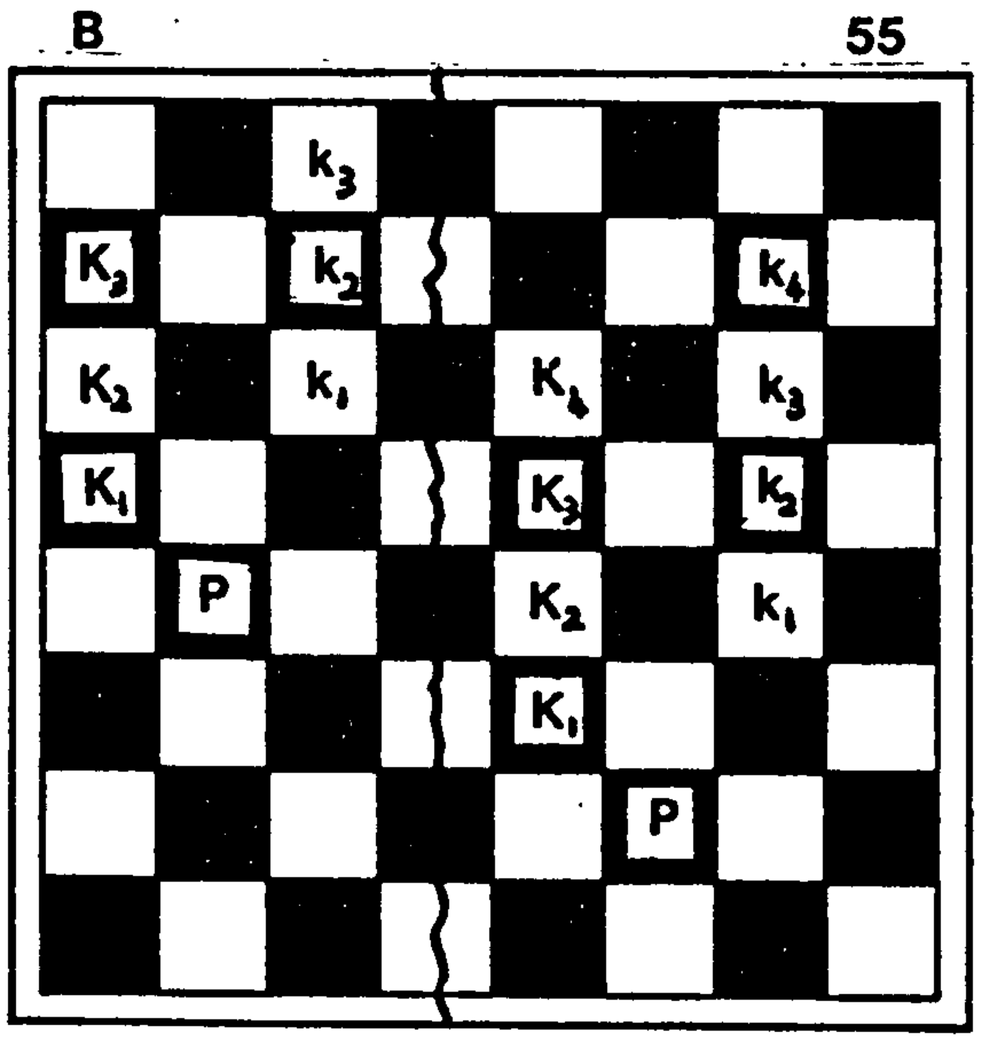
(Positions in classes 1, 2 and 3 are "terminal", i.e. the game is over, drawn in the first and second cases, won by White in the third.)

The form of the opposition referred to in rules 7, 12 and 13 is the "direct vertical opposition" with the White King below the Black. This can be defined as:

WK1 = BK1 AND BK2 = WK2 + 2

Pattern A and pattern B are shown in Figures 54 and 55, respectively, below.

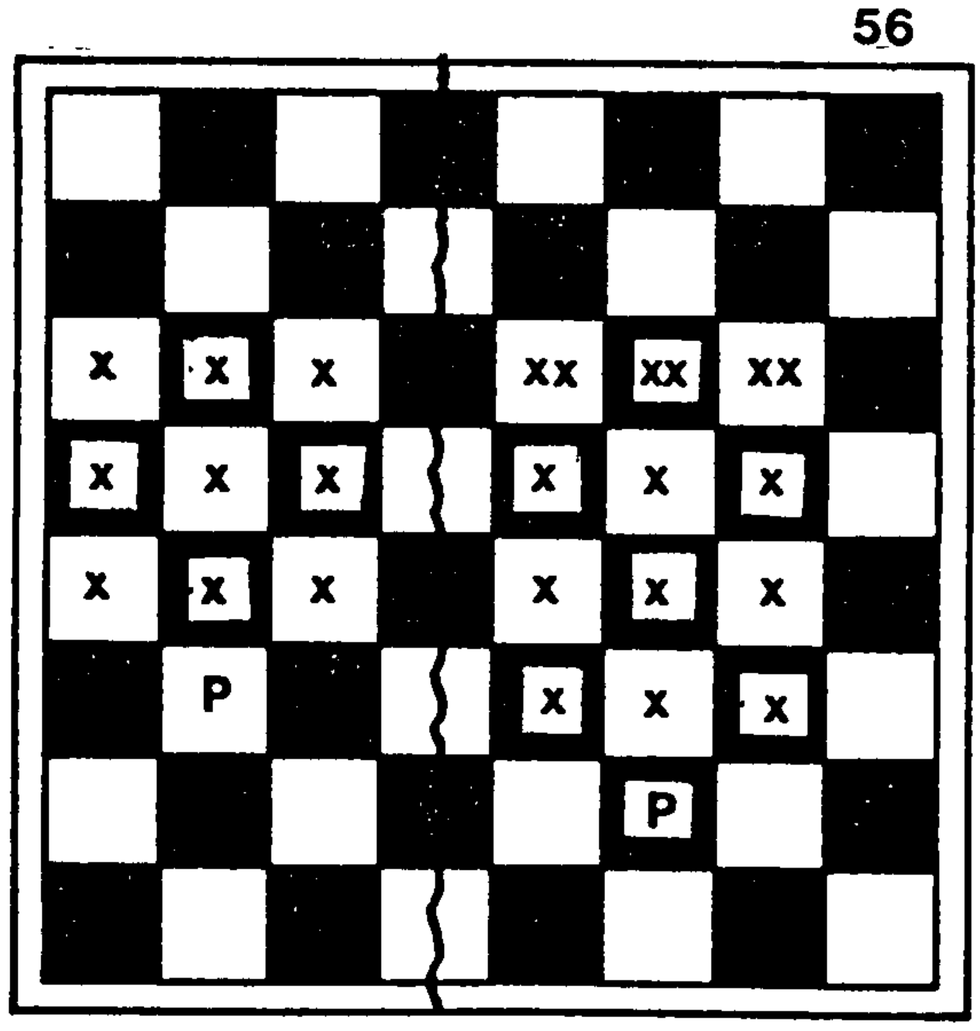




Here K_1 and k_1 are to be taken as one possible position of the two kings, K_2 and k_2 as another, etc. The Pawn may be on any square of the board and the positions which are mirror images of those shown (about the Pawn's file) are also included. Figure 55 shows the two possibilities for pattern B; K_4 and k_4 are only to be included when the Pawn is on the second rank.

The term "critical square" is used here to mean one of the squares in the neighbourhood in front of the Pawn, as shown in Figure 56.

56



Once again the Pawn may be on any square of the board and the two possibilities are shown; the fourth row of squares (marked XX) is included only when the Pawn is on the second rank. It will be seen from Table 10 that no distinction has been made between a Rook Pawn and any other Pawn, although textbooks generally consider a Rook Pawn to be a special case in this ending.

However, as the advice offered by textbooks seemed to be broadly similar in the two cases (subject to board edge restrictions), with the principal difference being not the play but the result of the game, it was decided to treat Rook Pawns in the same way as other Pawns at this stage.

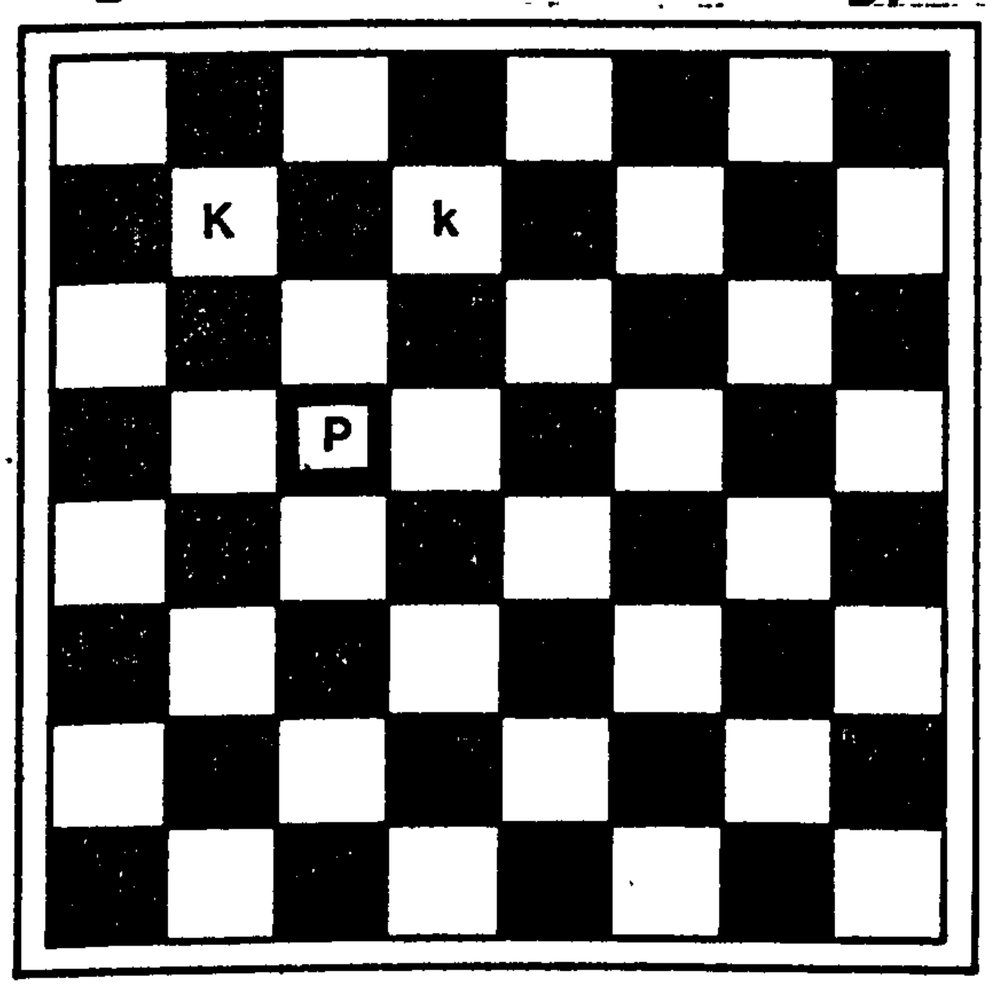
As will be seen in Section 10 this decision turned out to be an error. (Classes 9 and 10 cannot, of course, occur in the case of a Rook Pawn.)

Classes 4 to 14 are discussed below.

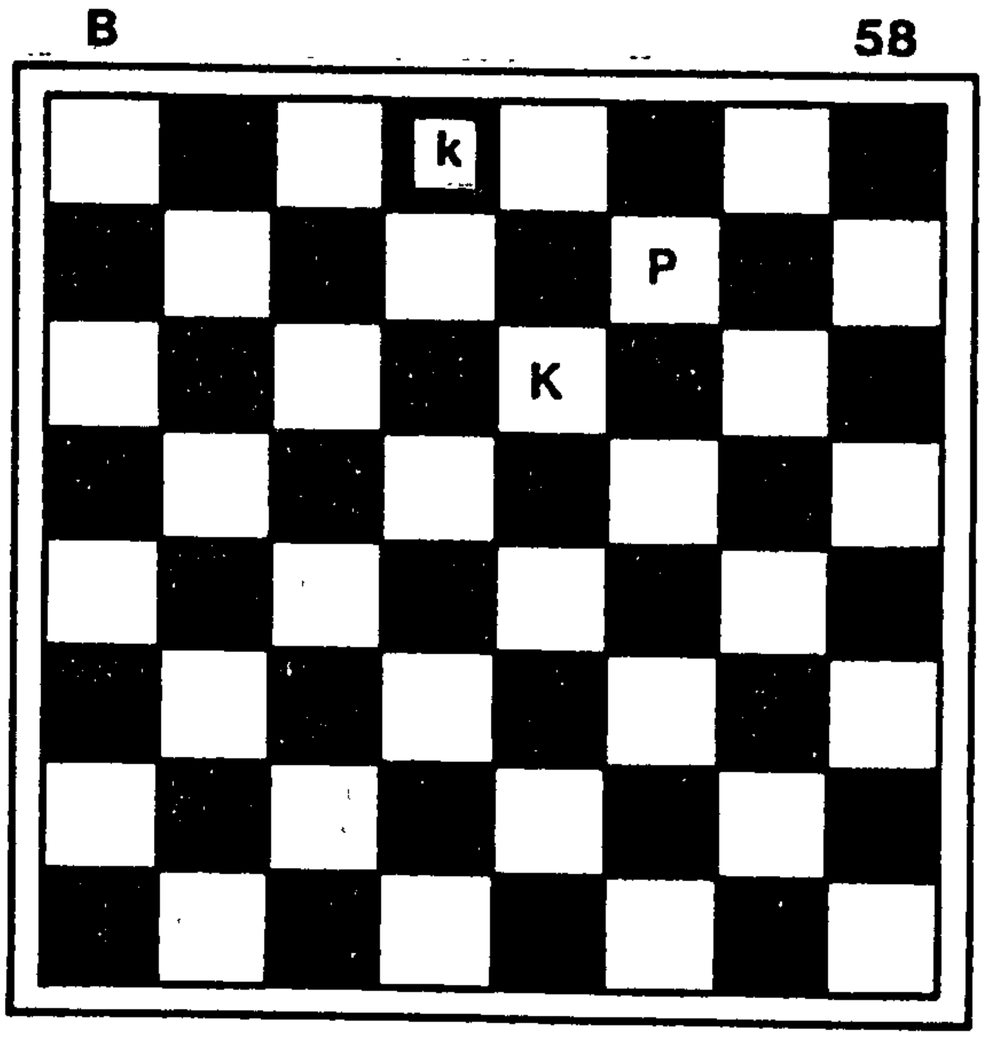
Class 4

This class contains positions (with Black to move) in which, against any play by Black, White can win simply by advancing his Pawn until it reaches the eighth rank (without making any King moves). Such positions include those where Black is outside and cannot immediately enter the "queening square" of the Pawn (for further details see, for example, Golombek (1954)) positions such as Figure 57 where the White King "dominates" all the squares in front of the Pawn and the small but important set of positions such as Figure 58 with the Pawn on the seventh rank. All positions in this class are theoretical wins for White.

B 57



B 58



The full definition used is as follows:

BK2 + 1 < WP2*

OR {abs (BK1-WP1)-1>8-WP2*}

OR {abs (WK1-WP1)=1 AND WP2=5 AND WK2=7}

OR {abs (WK1-WP1)=1 AND WP2>5 AND WK2>6}

OR {WP2=7 AND WK2=6 AND abs (WK1-WP1)<3

AND [WK1=WP1 OR (WK1<WP1 AND BK1<WP1) OR (WK1>WP1 AND BK1>WP1)]}

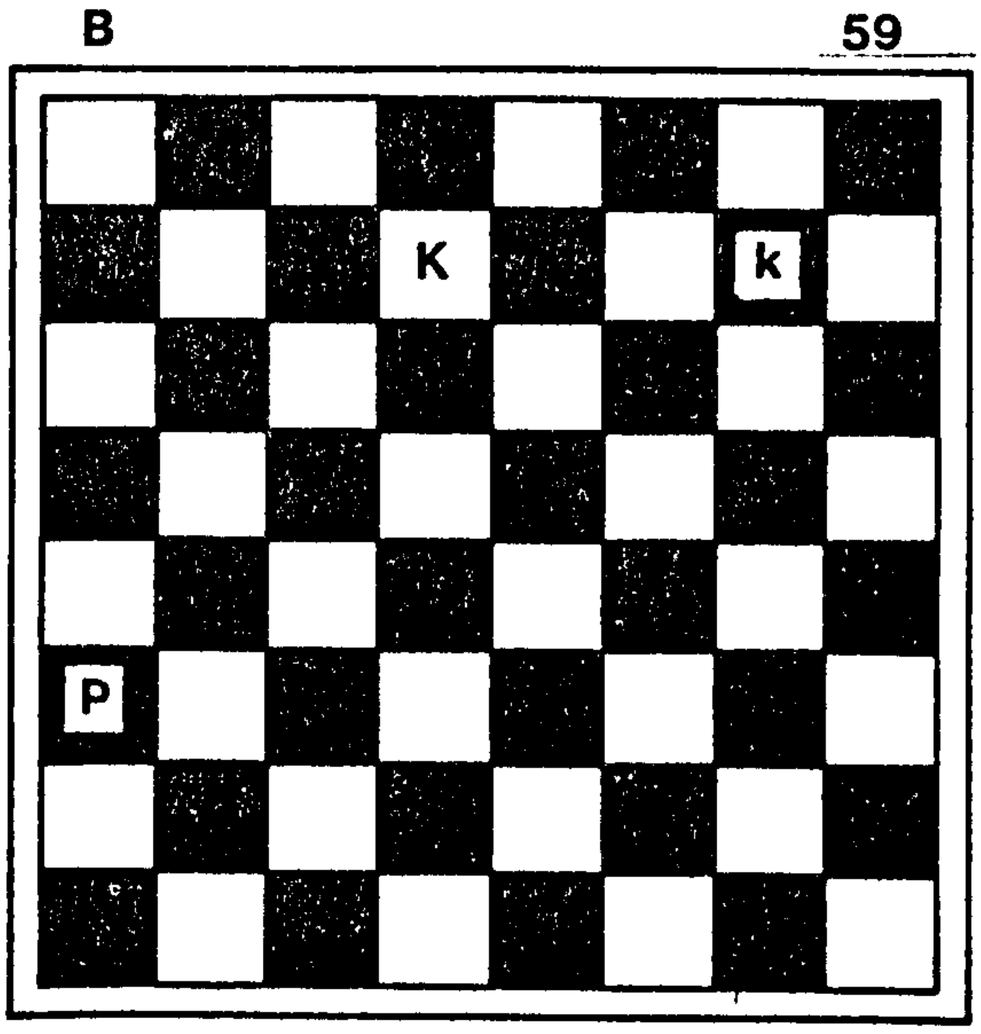
where WP2*=3 when WP2=2, and WP2 otherwise.

Although this definition is complex, it should be noted that there are less obvious positions than those previously described in which the Pawn can 'run', which are not included. For example, in Figure 59 Black to move cannot stop the Pawn.

Including all such positions in the definition of rule 4 is awkward and necessitates including a large number of "special cases" which do not appear to be particularly meaningful to the chess player, and they have therefore been excluded from membership of the class.

This is therefore an example of an equivalence class which can be specified simply in general terms (the Pawn can 'run') but where in practice it may be preferable to implement an approximation to the complete definition.

B 59



Class 5

The class is defined in its simplest form by the predicate.

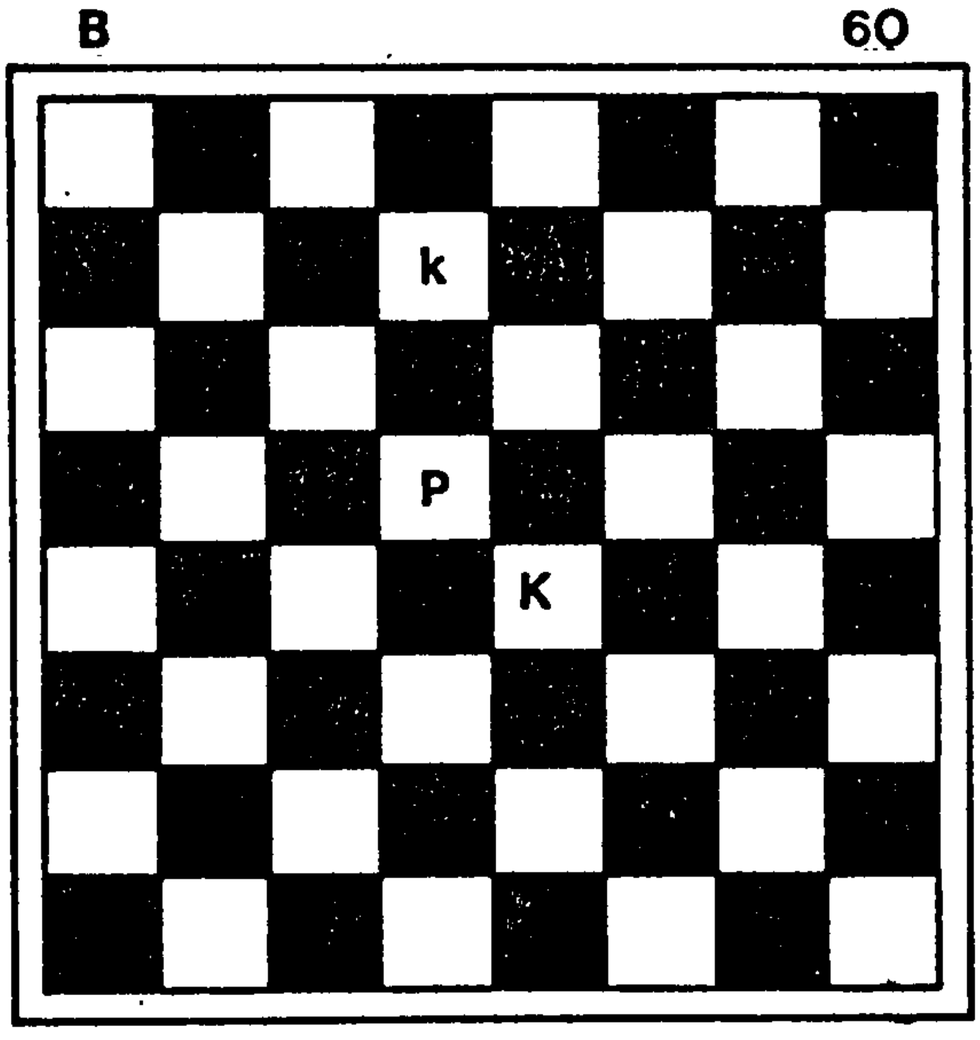
dist (BK1, BK2, WP1, WP2) < dist (WK1, WK2, WP1, WP2)

i.e. the block distance is less between the Black King and the Pawn than between the White King and the Pawn. The value of the left-hand side of this expression is increased by one when the Black King is diagonally ahead of the Pawn, since then an extra move is needed for Black to reach the Pawn. When this condition is satisfied it is to be expected that Black will be able to capture the Pawn and draw the game. (Although such a conclusion is justified in general, it will be seen in Section 10 that it is not invariably true when the Pawn is on the second rank and that a modification to the definition is needed in this case.)

Note that cases where in addition Black is outside and cannot immediately enter the queening square can be ignored here. These are handled by Class 4. In passing, it may be worth pointing out that in positions where the two block distances are equal (with Black to move) the Pawn sometimes but not always can be saved. For example, with Pawn on D4 and Black King on D6, a White King on B6 cannot save the Pawn, but a White King on F4 can.

Class 6

Class 6 contains positions where Black to move can immediately occupy the square in front of the Pawn, called by Nimzowitsch (1929) the "blockade" square. It can be proved fairly easily that in all such positions Black can draw the game.



For example, from Figure 60 play may continue

1.... K-D6; 2. K-D4, K-D7; 3. K-E5, K-E7;

4. P-D6ch, K-D7 5. K-D5, K-D8; 6. K-E6, K-E8;

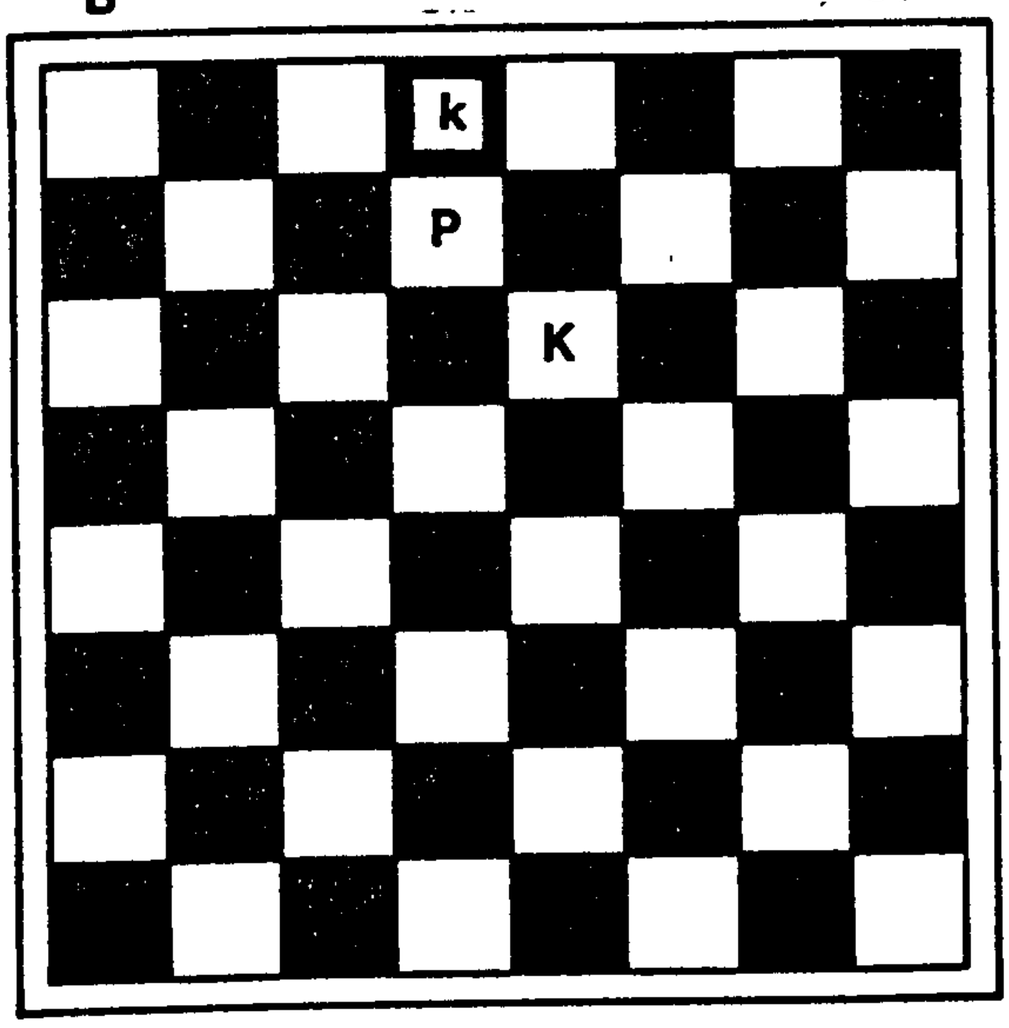
7. P-D7ch, K-D8; 8. K-D6 Stalemate.

Nimzowitsch calls the square <u>two</u> in front of the Pawn the "reserve blockade" square. Black should play to occupy the blockade square where possible and failing that the reserve blockade square or "opposition square" (two squares in front of the White King on the same file), in that order of preference. In the above example, the Black King's first three moves are to the blockade, reserve blockade and opposition squares respectively.

An apparent exception to the statement above that all positions in Class 6 are theoretical draws is provided by Figure 61 where Black to move loses (1.... K-C7; 2. K-E7 wins). However, here Black's King already occupies the blockade square and a simple analysis shows

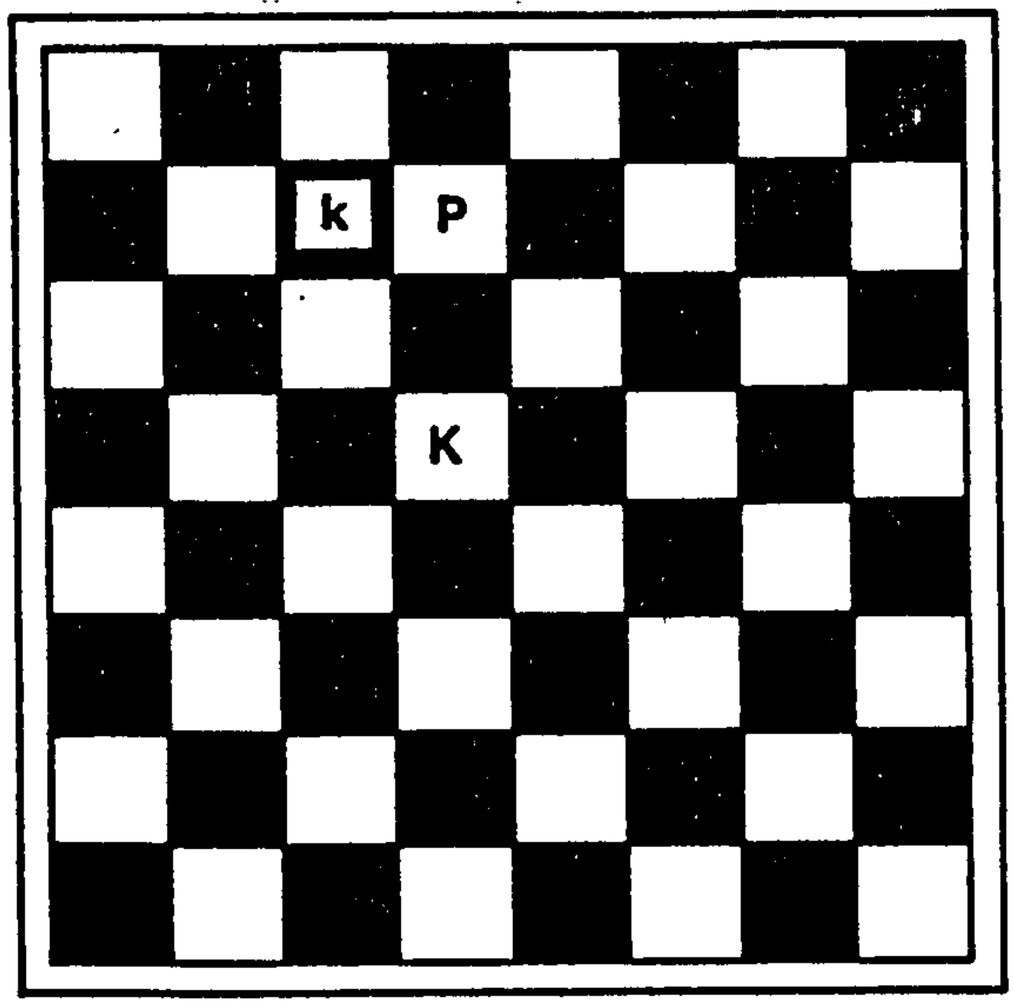
that either White's last move was P-D7, in which case when Black played to D8 it was not the blockade square, or the position must have arisen from an earlier position such as Figure 62 (after 1.... K-D8; 2. K-E6). However, in Figure 62 Black can, of course, capture the Pawn instead. The position is, in fact, a member of Class 1 not Class 6.

B



61

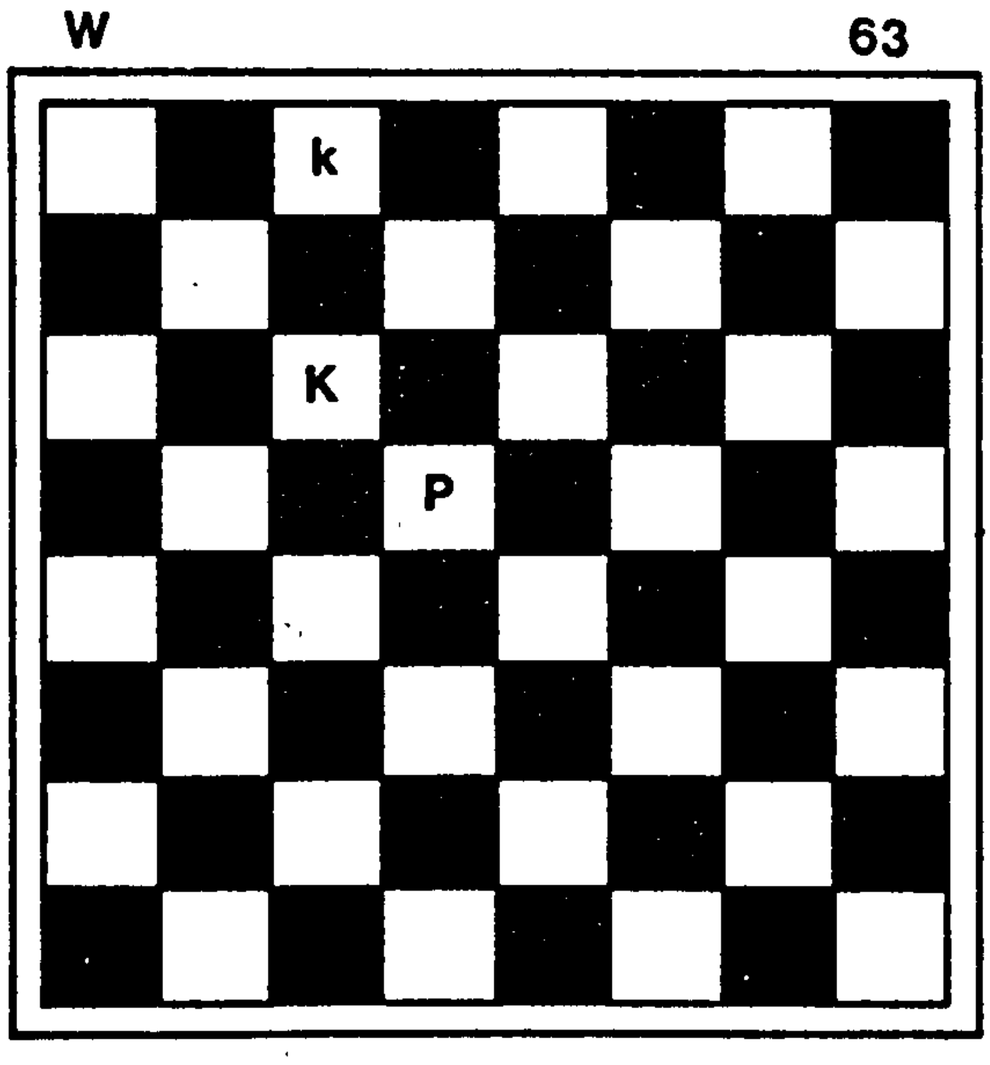
B



62

Class 7

This class of positions where White's King is on the blockade square and Black can take the opposition is included with a low class value in order to direct White's play in certain critical situations.



In Figure 63, White should not play 1. K-D6 although this move maintains his winning advantage. Black will then play K-D8, leaving White with no better than 2. K-C6. Black can then return to Figure 63 by K-C8. In order to convert his advantage to victory, White should play 1. P-D6, K-D8; 2. P-D7, K-E7; 3. K-C7 etc.

Including Class 7 in the algorithm will ensure that White will reject the position arising after 1. K-D6 from consideration, as a member of a low ranked class.

Class 8

This class of positions where White is two or more files closer to the Pawn than Black (and not below the rank of the Pawn) is also included to handle particular important situations which can occur, this time

by specifying a class of extremely favourable positions, with a high class value.

W — 64

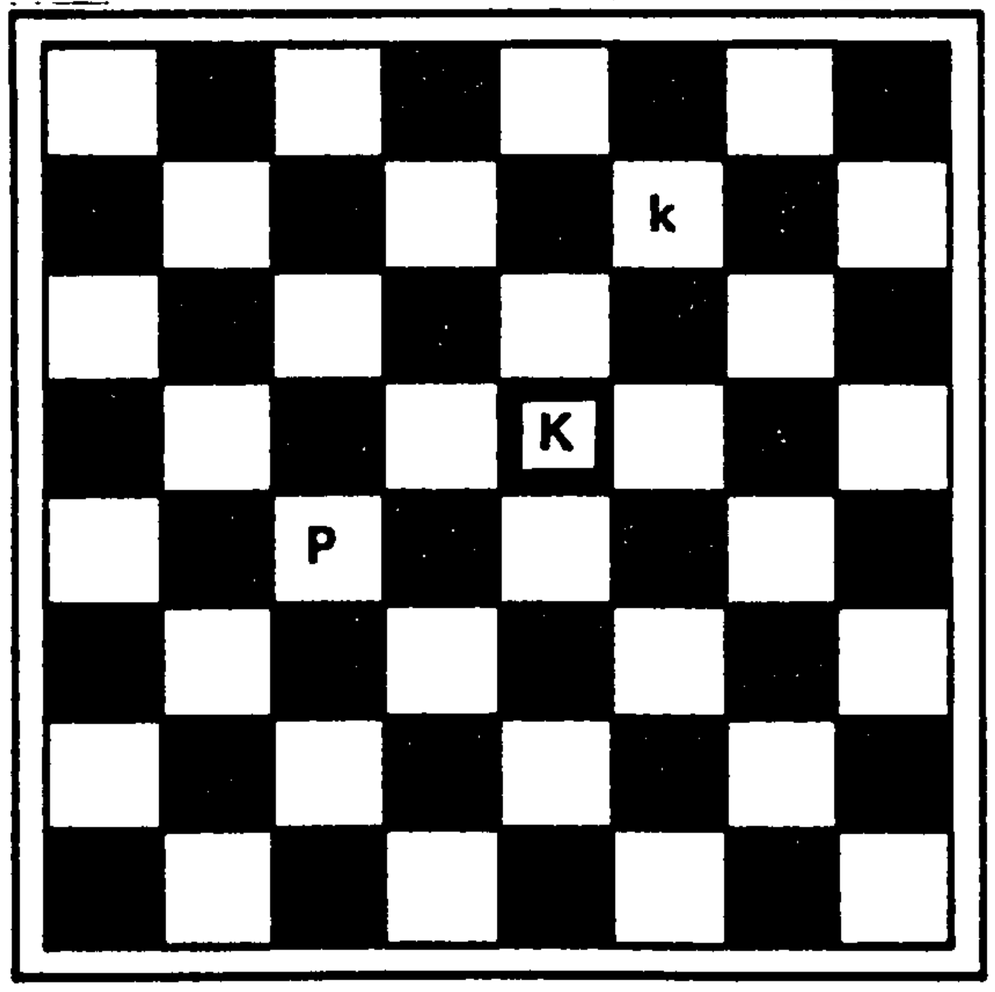
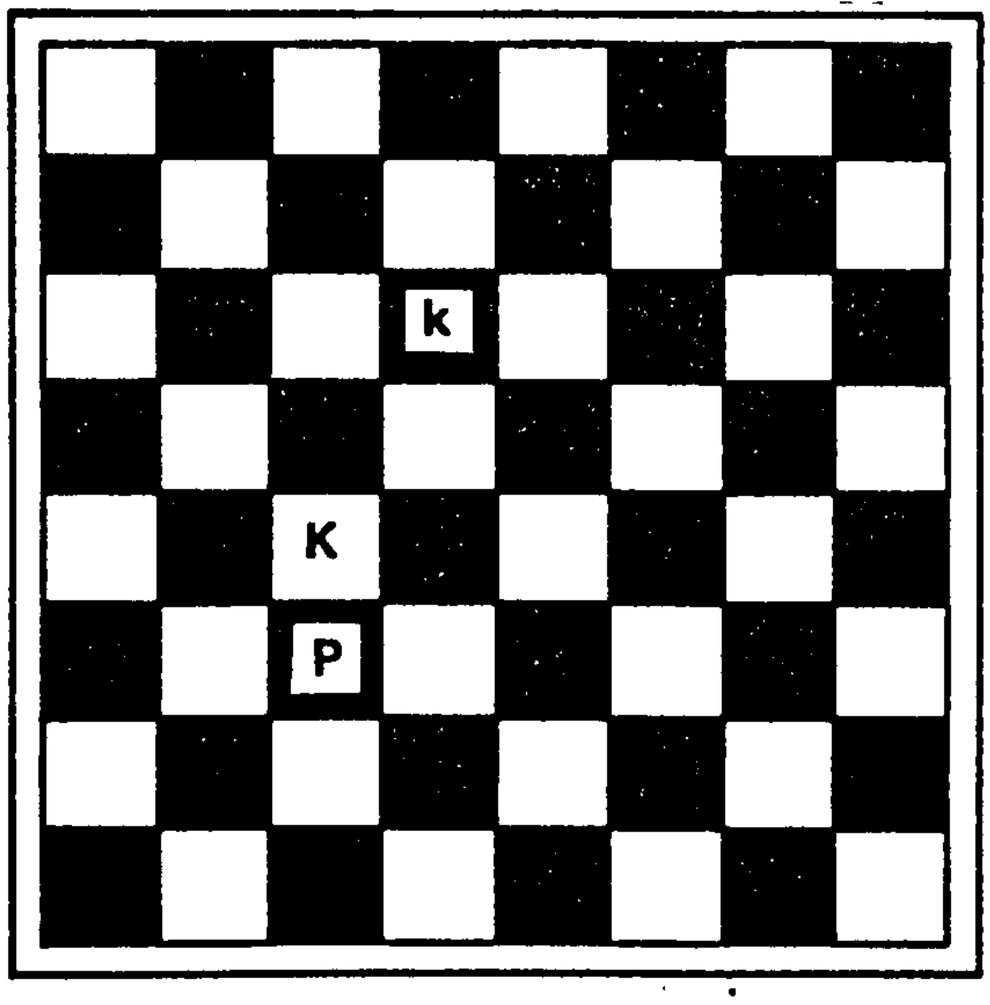


Figure 64 shows a fairly common situation. White should not now play 1. K-F5 taking the opposition, since then after Black's reply K-E7, he must play 2. K-E5, whereupon Black, if he wishes, may return to Figure 64 by K-F7. Figure 64 is the critical position at which White must break out of this cycle. He should play 1. K-D6, moving two files closer to the Pawn than Black and winning easily. The position arising after 1. K-D6 is a member of class 8 and its high class value ensures that K-D6 will be chosen by the move finding algorithm.

Classes 9 and 10

These two classes are included to handle positions similar to Figure 65.

W 65



In this position (where Black will typically have just moved from C6 to D6) White should not take the opposition by K-D4, but "sidestep" with 1. K-B5 (the resulting position being a member of Class 10). If Black now plays K-D7, White should reply K-B6 with a Class 10 position once again.

If Black instead chooses 1.... K-D5, White should play 2. P-C4ch giving a position in Class 9 (pattern A). This example shows the significance of including Class 9 as well as Class 10. Without it White would play 2. K-B4 once again attaining pattern B.

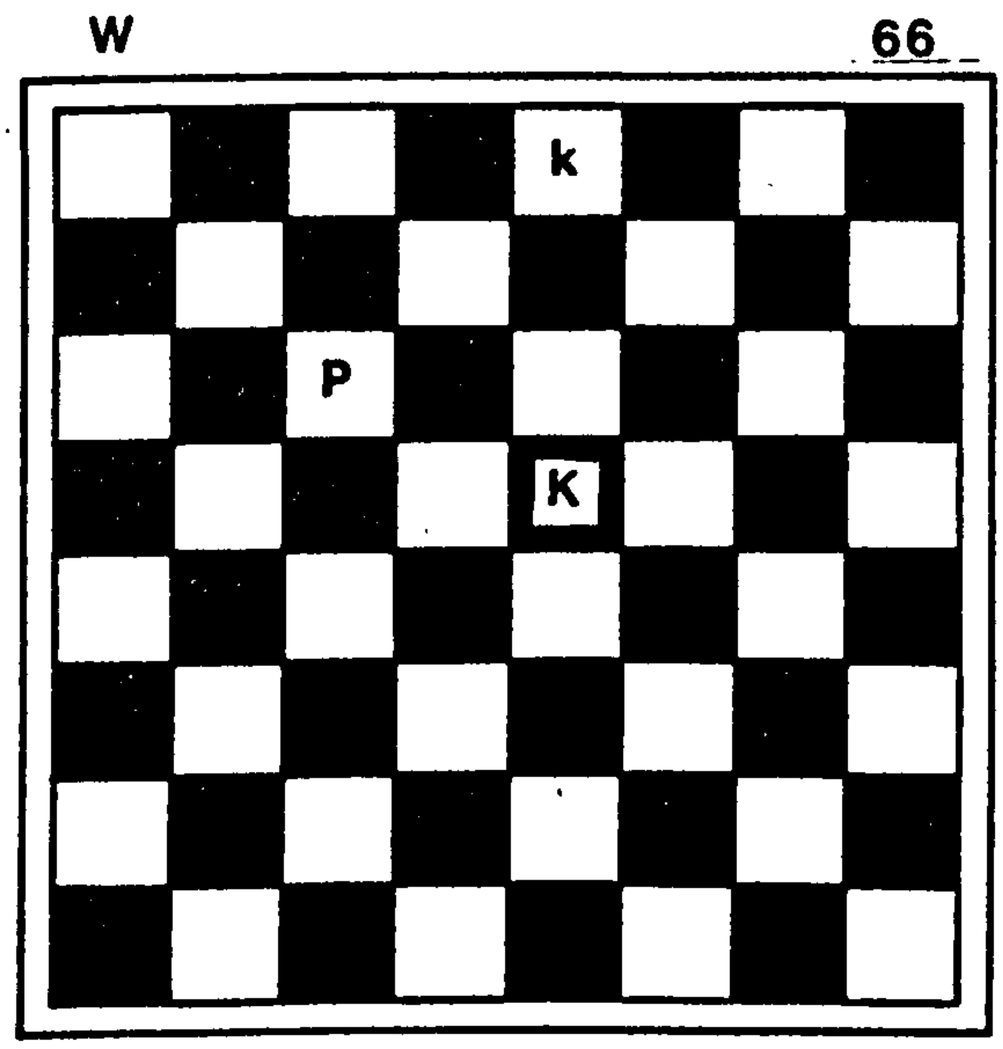
Class 11

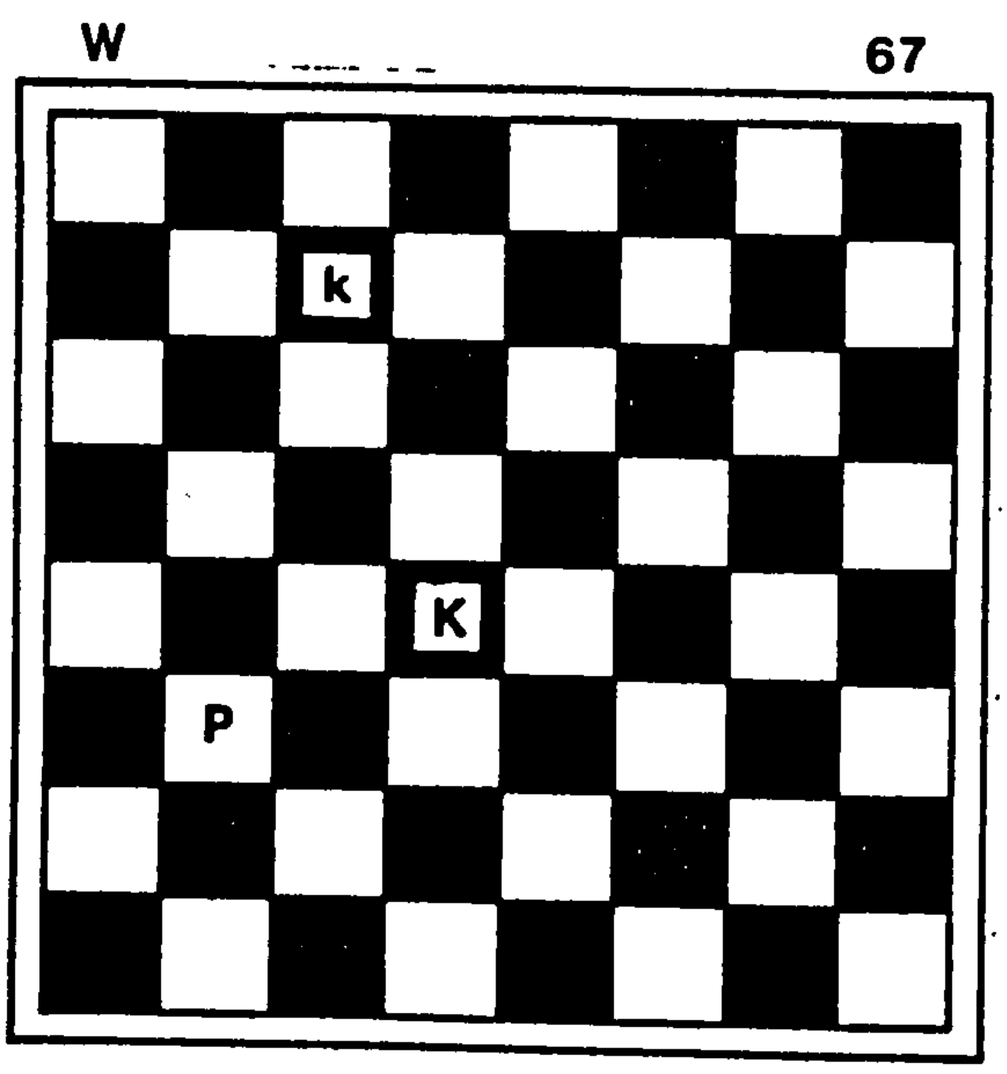
After 1. K-B5, K-C7 in Figure 65, White has only one move to win, 2. K-C5. The existence of Class 11 ensures that this move will be chosen, although such an objective could in this case be obtained by other means (e.g. using Class 13).

In general, White wins whenever his King is on the Pawn's file and in front of the Pawn simply by advancing the Pawn as rapidly as possible (assuming that Black is not close enough to capture the Pawn). The only difficulty is that he should avoid advancing the Pawn to immediately below the King when Black can take the opposition. Such cases are taken care of by Class 7, as has already been described.

Classes 12 and 13

All members of these classes are positions which are won for White except in the case of a Rook pawn and (for class 13 only) positions where the White King and Pawn are on the same rank. This latter type of position is only won when both pieces are on the sixth rank. For example, in Figure 66 White wins by 1. K-E6 (e.g. 1....K-D8; 2. K-D6, K-C8; 3. P-C7 etc.). The alternative 1. K-D6 only draws (e.g. 1.... K-D8; 2. P-C7ch, K-C8; 3. K-C6 stalemate). In this case the position after 1. K-E6 belongs to Class 13.





Class 12 is included to ensure correct play in positions such as Figure 67. White here wins by 1. K-C5, K-B7; 2. K-B5. If Black now plays K-C7 then 3. K-A6 sidestepping as before (pattern B).

The positions after White's three moves in this sequence are members of classes 12, 11 and 10, respectively.

Class 14

All positions belonging to this class are won for White, except for some positions with a Rook Pawn and those where Black can immediately take the opposition, with the White King on the fifth rank or below and only one rank in front of the Pawn. These positions are drawn.

8.2 The associated functions

There are seven different associated functions used for this form of the algorithm.

Table 11 Associated functions

Function	Value
1	WP2
2	8-dist (WK1, WK2, WP1, WP2)
3	8-min {abs (WK1-WP1), abs (WK2-WP2)}
4	8-abs (WK1-WP1)
5	abs (WK1-WP1)
6	WK2-WP2
7	WK2

where dist (WK1, WK2, WP1, WP2) is defined as max {abs (WK1-WP1), abs (WK2-WP2)

8.3 The value table

The value table consists of 15 rows and 4 columns, the first column holding the class value and the remainder the associated functions f_1, f_2 and f_3. There are no more than three functions for any of the classes.

Table 12 Value table

Class	Class Value	f_1 f_2 f_3
1	1	2 3 0
2	2	0 0 0
3	15	0 0 0
4	14	1 0 0
5	3	2 3 0
6	4	1 2 3
7	5	0 0 0
8	13	1 4 7
9	12	1 0 0
10	11	1 0 0
11	10	1 0 0
12	9	0 0 0
13	8	1 0 0
14	7	1 6 5
15	6	2 3 7

As in the case of King and Rook against King, the relative ordering of class values corresponds to an ordering of White's objectives, thus Class 10 is more valuable to White then Class 12 since from such positions he controls the squares in front of the Pawn and thus ensures its advance.

9. Refining the algorithm: I - background and rationale

The algorithm described in the previous section was used as the starting point for experimentation to determine how readily algorithms based on the "equivalence class" model could be modified to play perfectly, given external knowledge of the best move in every situation for a particular endgame.

In terms of the objectives given in Section 1, this experimentation was intended partly as a further demonstration of the use of the model described in Section 2, but primarily as an example of the modifiability of algorithms under a different kind of testing from that described in Section 6 and as a means of isolating some of the features which would be required in a self-improving system capable of refining its own performance.

The feasibility of constructing a self-improving system is discussed in Section 11 in the light of this experimentation. In general, perfect external knowledge is not available and the objective of perfect play must necessarily be modified accordingly.

The usual situation is that a substantial amount of information is known about some specific endgame, for example analyses in textbooks and games played by expert players. The aim is then to produce a program giving the best "fit" to the available information, with the complication that in many cases even the strongest human players do not play optimally (in the sense of choosing the shortest winning move in every position, etc.) and the strategies given in different textbooks or chosen by different players may be conflicting. The possibility of genuine mistakes also

cannot be ruled out.

In section 6 an example was given of improving the King and Rook against King algorithm against a background of imperfect knowledge, the skill of the human opponents being unknown (thus the program may on occasions have won despite playing badly). Such investigations are necessarily unsystematic. It is always possible that the next game played would have revealed new difficulties.

One advantage of using a store of absolute knowledge of optimal play, in the few cases where this is available, is that it enables the performance of a program to be measured on an objective scale and thus be capable of comparison with both other programs and human chess players.

Further, improving a program to play perfectly for a given set of positions may provide valuable information on the tools needed in the more general case of only imperfect knowledge, when the objective is play which, in some sense, is <u>better</u> than the knowledge available.

The database to be used is one of two set up by Clarke (1975) for simple endgames, once again King and Rook against King and King and Pawn against King. The method used is to back-up a tree of variations from terminal positions (for which the result, win or draw, is known "by definition") to earlier positions and thus to determine the result against best play by the opponent in each case and also the "depth" of each positon, defined as the number of moves from the nearest terminal position, given best play on both sides. Given these depths the best move, or all equally good "best" moves, in any position can be found; since the depth of the resulting position will be reduced by one in each case with the same overall result (win or draw). The order of testing of the rules has once again been chosen to simplify the definitions where possible but is otherwise arbitrary.

For the experimentation to be described in the next section, use was made of the database for King and Pawn against King. This was chosen in preference to King and Rook against King because of its greater practical interest for chessplayers, in particular because of the large number of positions which exist where White (the side with the Pawn) can only win by a very precise sequence of play. The database contains only positions with the Pawn in the left-most four files of the board. All other positions are, of course, equivalent to these by symmetry.

Since the aim was to modify the original algorithm given in Section 8 to play perfectly by "manual" changes only, a relatively small sample of significant positions was chosen for analysis. The positions chosen were all those for which White has one and only one winning move, against best play by Black.

These positions can be identified by generating each of the possible successor positions with Black to move. One of these will be a loss for Black with a depth one less than in the original position, the others will all be draws.

Cases in which White can win by advancing his Pawn and continuing to advance it every move until it reaches the eighth rank were excluded. (These positions can easily be determined since the depth is equal to the number of Pawn moves required to reach the eighth rank.)

Positions where the Black King was more than one rank or file outside the queening square were also excluded, since even in the worst case, where White blocks his own Pawn, he can simply move his King off the Pawn's file and the Pawn can then 'run'.

There were 1733 positions remaining and these were set up as a test file, together with White's best move in each position, in suitable coded form.

It was felt that 1733 positions constituted a manageable sample from the full set of 81662 legal positions with White to move and the Pawn on files A to D, ranks 2 to 7 inclusive, held on the database. To improve the algorithm "manually" to play perfectly for all these positions is likely to be a much more lengthy process but would not necessarily provide any further insights. Of these 81662 legal positions, only 62,480 are theoretical wins for White (Bramer, 1977*).

Analysing these won positions further shows that 47223 of them are "trivial" in the sense that the Black King is outside the queening square and the White King is not in front of the Pawn. The presence of class 4 in the algorithm will certainly ensure that all these are handled correctly.

There are therefore only 15257 non-trivial won positions. Thus a sample of 1733 positions is a fairly large proportion of all the "interesting" won positions.

In the description which follows, White's best move in each position on the test file will be referred to as the 'file move'. Thus in every case the file move is the only one for White to win, all others leading only to draws against best play by Black.

---

* Reprinted at the end of this thesis.

The method used to test the performance of the algorithm was 'exception reporting'. A program embodying the algorithm described in Section 8 was set up and for each position on the test file the move chosen by the program (the 'program move') was calculated and compared with the 'file move'. Those cases where there was a difference were printed out in tabular form. These 'exceptions' were then inspected, an appropriate change made to the algorithm and a new table produced.

With the initial version of the algorithm, there were 193 exceptions, which were completely eliminated after a fairly lengthy but straightforward series of amendments, which are described in detail in the following section. The final version of the algorithm therefore plays perfectly in all of the test positions on file, but since these are only a sample of all possible positions for the King and Pawn against King endgame, it cannot automatically be assumed that the program will play perfectly in all other cases. To modify the program further, if necessary, to play perfectly in every position might involve a substantial amount of time to perform manually, but would be quite feasible if the processes involved could be automated along the lines suggested in Section 11. The overall level of performance of the final algorithm will be discussed in Section 10.5.

The process of program improvement involved some changes to the associated functions used in the algorithm and more importantly an increase in the number of equivalence classes from 15 to 24. As will be seen, however, the final number of classes can be reduced to as few as 14 if required, without any reduction in performance.

The information produced by the exception reporting system enables possible simplifications to the algorithm (such as the combination of two or more equivalence classes) to be readily detected (at any stage,

not necessarily at the end of the development process), and gives an indication of the importance of each of the component parts, for example how many positions could possibly be affected by the deletion of an equivalence class.

Moreover, the method of exception reporting enables simplifications, generalizations etc. which are not provably justified to be investigated on an empirical basis.

The experimentation is described in detail in the following section.

10. Refining the algorithm: II - experimentation

This section consists of a description of the modifications which were made to the King and Pawn against King algorithm and the judgements and decisions on which these were based, together with some general discussion of the processes involved.

10.1 The initial algorithm

The table below shows the class value and associated functions for each of the 15 classes in the initial form of the algorithm described in Section 8. In this and subsequent similar tables the classes will be given in the order in which the corresponding rules are tested (1,2,3,...,15 initially).

Table 13 The initial algorithm

Class	Value	Associated functions
1	10	2,3,0
2	20	0,0,0
3	150	0,0,0
4	140	1,0,0
5	30	2,3,0
6	40	1,2,3
7	50	0,0,0
8	130	1,4,7
9	120	1,0,0
10	110	1,0,0
11	100	1,0,0
12	90	0,0,0
13	80	1,0,0
14	70	1,6,5
15	60	2,3,7

Note that the class values given in Section 8 have all been multiplied by ten here, to allow for possible insertions. Of the

1733 test positions on file, 193 exceptions (11%) were identified with this form of the algorithm and these are given in Table 14. The entries in the table should be interpreted as follows

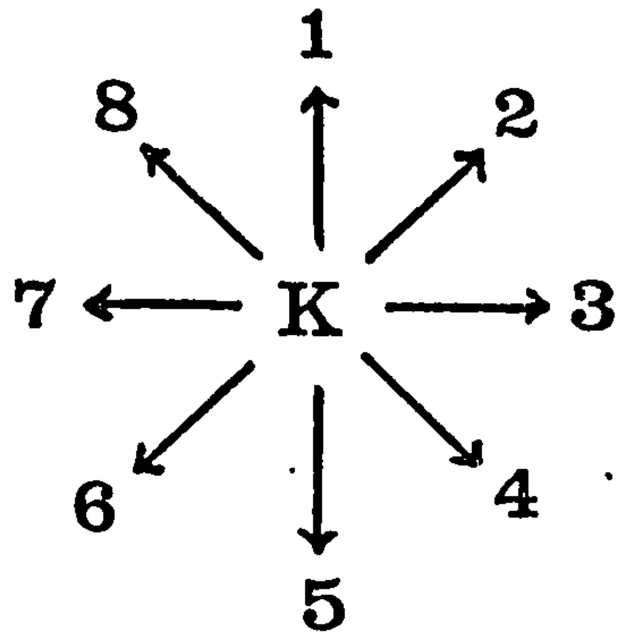
(1) REF - a reference number from 1 to 1733.

(2) POSITION - the initial position, in coded form. The six digits indicate, from left to right, the White King's file and rank co-ordinates followed by those of the Black King and the White Pawn, i.e. WK1,WK2, BK1,BK2, WP1 and WP2, respectively.

(3) MOVEF - the 'file move' in a given position with White to move, i.e. the best move as held on the test file.

(4) MOVEP - the 'program move' in a given position with White to move, i.e. the move selected by the current form of the algorithm.

For both these moves the code used is to represent King moves by an integer from 1 to 8, according to the following scheme.



A single pawn move is represented by 9, and a double move by 10.

(5) CLASSF - the equivalence class of which the position arising after playing MOVEF from a given initial position is a member.

(6) CLASSP - the equivalence class of which the position arising after playing MOVEP from a given initial position is a member.

(7) SEL - 'selection level'. This indicates the 'level' at which the program move was selected in preference to the one considered next most favourable.

Selection by class value alone (i.e. the program move was the only one in the highest ranked class) is indicated by 1. Selection by means of the functions associated with the most favourable class is shown by 2, if only the first function is needed, 3 if the second function is needed and 4 if the third function is needed.

The letter "T" indicates that an arbitrary choice was made, to resolve a tie.

TABLE 14 EXCEPTIONS WITH THE INITIAL ALGORITHM

REF	POSITION	MOVE F	MOVE P	CLASS F	CLASS P	SEL
2	381742	6	10	15	15	2
16	872842	6	10	15	15	4
26	772642	6	10	15	15	2
44	873842	6	10	15	15	4
46	163742	4	10	15	15	4
50	763742	6	10	15	15	4
75	124742	2	3	15	15	3
77	724742	8	7	15	15	3
92	875842	6	10	15	15	4
94	165742	4	10	15	15	4
98	765742	6	10	15	15	4
125	416742	8	2	15	15	T
128	176642	4	10	15	15	2
135	416542	8	2	15	15	T
146	756342	7	10	5	5	3
161	677442	6	10	5	15	1
169	788442	6	10	5	15	1
199	132843	2	3	15	15	3
230	733843	8	7	15	15	3
259	734843	8	7	15	15	3
278	735843	8	7	15	15	3
307	736843	8	7	15	15	3
316	426643	8	2	15	15	T
336	837843	8	7	15	15	3
476	436744	8	2	15	15	T
524	141745	2	1	15	13	1
579	446845	8	2	15	15	T
584	656745	9	7	15	15	3
650	771832	6	10	15	15	4
671	772832	6	10	15	15	4
675	662732	6	10	15	15	4
700	623732	8	7	15	15	3
715	774832	6	10	15	15	4
719	664732	6	10	15	15	4
744	315732	8	2	15	15	T
746	315632	8	2	15	15	T
754	315432	8	2	15	15	T
762	726732	8	7	15	15	3
778	827732	8	7	15	15	3
786	687432	6	10	5	15	1
815	632833	8	7	15	15	3
859	634833	8	7	15	15	3
887	635833	8	7	15	15	3
892	325633	8	2	15	15	T
906	736833	8	7	15	15	3
1021	335834	8	2	15	15	T
1029	335634	8	2	15	15	T
1112	345835	8	2	15	15	T
1116	555735	9	7	15	15	3
1173	671822	6	10	15	15	4
1177	561722	6	10	15	15	4
1199	522722	8	7	15	15	3
1212	673822	6	10	15	15	4
1215	563722	6	10	15	15	4
1236	214722	8	2	15	15	T
1239	214622	8	2	15	15	T
1249	214422	8	2	15	15	T
1258	625722	8	7	15	15	3
1274	726722	8	7	15	15	3
1282	586422	6	10	5	15	1
1287	827722	8	7	15	15	3
1318	532823	8	7	15	15	3
1352	224823	8	2	15	15	T
1355	224723	8	2	15	15	T
1365	224523	8	2	15	15	T
1397	736823	8	7	15	15	3
1463	234824	8	2	15	15	T
1473	234624	8	2	15	15	T
1542	244725	8	2	15	15	T
1552	545725	8	1	15	13	1
1592	453312	7	10	5	5	3
1596	264812	1	10	8	8	2
1599	174712	3	10	8	8	2
1601	264712	1	10	8	8	2
1603	154612	2	10	8	8	2
1605	174612	4	10	8	3	2
1607	144512	2	9	8	8	2
1609	244512	1	10	8	8	2
1612	174412	4	10	5	8	1
1614	374412	6	10	5	15	1
1622	485512	6	10	5	15	1
1624	485312	6	10	5	15	1
1645	164813	2	9	8	8	2
1648	164713	2	9	8	8	2
1650	184713	4	9	8	8	2
1652	284713	5	9	8	8	2
1654	164613	3	9	8	8	2
1656	254613	1	9	8	8	2
1659	174513	4	9	8	8	2
1662	174413	4	9	8	8	2
1689	153414	2	1	14	8	1
1693	264814	1	9	8	8	2
1696	174714	3	9	8	8	2
1698	264714	1	9	8	8	2
1700	184614	4	9	8	8	2
1703	184514	4	9	8	8	2
1723	254615	1	8	8	8	3
10	271442	4	10	5	15	1
17	122742	2	3	15	15	3
37	152342	3	10	5	5	3
45	123742	2	3	15	15	3
49	723742	8	7	15	15	3
74	874842	6	10	15	15	4
76	164742	4	10	15	15	4
78	764742	6	10	15	15	4
93	125742	2	3	15	15	3
97	725742	8	7	15	15	3
122	876842	6	10	15	15	4
126	726742	8	7	15	15	3
129	416642	8	2	15	15	T
141	416442	8	2	15	15	T
151	827742	8	7	15	15	3
167	788542	6	10	5	15	1
170	788342	6	10	5	15	1
228	133843	2	3	15	15	3
257	134843	2	3	15	15	3
276	135843	2	3	15	15	3
306	426843	8	2	15	15	T
310	426743	8	2	15	15	T
322	426543	8	2	15	15	T
470	436844	8	2	15	15	T
482	436644	8	2	15	15	T
537	252745	9	3	15	15	3
583	446745	8	2	15	15	T
596	747745	8	1	15	13	1
657	671632	6	10	15	15	2
674	622732	8	7	15	15	3
699	773832	6	10	15	15	4
701	663732	6	10	15	15	4
718	624732	8	7	15	15	3
743	775832	6	10	15	15	4
745	625732	8	7	15	15	3
750	315532	8	2	15	15	T
757	655332	7	10	5	5	3
772	576432	6	10	5	15	1
782	687532	6	10	5	15	1
787	687332	6	10	5	15	1
842	633833	8	7	15	15	3
886	325833	8	2	15	15	T
888	325733	8	2	15	15	T
896	325533	8	2	15	15	T
927	837833	8	7	15	15	3
1025	335734	8	2	15	15	T
1071	151735	9	3	15	15	3
1115	345735	8	2	15	15	T
1124	646735	8	1	15	13	1
1176	521722	8	7	15	15	3
1198	672822	6	10	15	15	4
1200	562722	6	10	15	15	4
1214	523722	8	7	15	15	3
1234	674822	6	10	15	15	4
1237	524722	8	7	15	15	3
1244	214522	8	2	15	15	T
1253	554322	7	10	5	5	3
1268	475422	6	10	5	15	1
1278	586522	6	10	5	15	1
1283	586322	6	10	5	15	1
1295	531823	8	7	15	15	3
1332	533823	8	7	15	15	3
1353	534823	8	7	15	15	3
1360	224623	8	2	15	15	T
1376	635823	8	7	15	15	3
1413	837823	8	7	15	15	3
1468	234724	8	2	15	15	T
1538	244825	8	2	15	15	T
1543	454725	9	7	15	15	3
1591	143312	2	9	5	8	2
1595	164812	2	10	8	8	2
1598	164712	2	10	8	8	2
1600	184712	4	10	8	8	2
1602	284712	5	10	8	8	2
1604	164612	3	10	8	8	2
1606	254612	1	10	8	8	2
1608	174512	4	10	8	8	2
1611	134412	2	1	8	8	3
1613	234412	1	9	8	8	2
1615	224312	1	8	8	8	3
1623	485412	6	10	5	15	1
1641	143313	2	1	14	8	1
1646	264813	1	9	8	8	2
1649	174713	3	9	8	8	2
1651	264713	1	9	8	8	2
1653	154613	2	9	8	8	2
1655	174613	4	9	8	8	2
1658	144513	2	1	8	8	3
1660	244513	1	9	8	8	2
1663	234413	1	8	8	8	3
1692	164814	2	9	8	8	2
1695	164714	2	9	8	8	2
1697	184714	4	9	8	8	2
1699	154614	2	1	8	8	3
1701	254614	1	9	8	8	2
1704	244514	1	8	8	8	3

The complete set of exceptions is summarised in the table below.

Table 15 Summary of exceptions: initial algorithm

CLASSP	CLASSF	Number of Exceptions	Selection Level 2	3	4	Ties
5	5	5	0	5	0	0
8	5	2				
8	8	45	38	7	0	0
8	14	2				
13	15	4				
15	5	17				
15	15	118	4	49	26	39

In this and subsequent similar tables, the selection level is only given in the cases where CLASSP and CLASSF are equal. In such cases, the "second best move" (as judged by the move-finding algorithm) must also give a position in the same equivalence class.

The "second best move" may, in fact, be MOVEF. If it is not, the only possibility is that the position arising after MOVEP was chosen from three or more positions in the same equivalence class, by means of associated functions. Positions where CLASSP and CLASSF are equal would seem to indicate that the program has made the correct choice of best available equivalence class but has failed to discriminate appropriately within that class. It is likely therefore that some change to the associated functions for that class is required.

When CLASSP and CLASSF are different, there are a number of possibilities. It may be that the rankings of CLASSP and CLASSF should be interchanged but it is more probable that the position arising after MOVEP is being ranked too high or, alternatively, that the position

arising after MOVEF is being ranked too low. It may therefore be necessary to amend the definitions of CLASSP or CLASSF (or both) or to introduce a new equivalence class or classes. The selection level is of little significance in these cases and is therefore omitted from the tables. This brief analysis ignores possible inter-relations between the exceptions (e.g. in three rows of Table 15, CLASSP is 8) and the necessity to consider the exceptions in the context of the correctly handled positions, which provide support for the existing components of the algorithm to a greater or lesser extent.

Table 16 below shows a complete breakdown for each equivalence class of the number of successor positions belonging to that class (note that there may be up to ten successor positions for each of the positions on the test file), the number of times a member of that class was selected as best (a total of 1733 for all classes) and the selection levels for this latter number of positions. In this and similar subsequent tables, only classes of which at least one successor position is a member (whether selected as best or otherwise) are included. Thus class 3 (Pawn on eighth rank, not subject to capture) will never be included. The selection criteria for the test file ensure that there are no cases where White can immediately promote his pawn and win.

Table 16 Selection Table : Initial Algorithm

CLASS	TOTAL	SELECTED	Selection Level 1	2	3	4	TIES
1	1367	0	0	0	0	0	0
2	10	0	0	0	0	0	0
4	179	179	179	0	0	0	0
5	2448	15	0	0	5	0	10
6	652	0	0	0	0	0	0
7	84	0	0	0	0	0	0
8	441	223	161	40	7	15	0
9	45	45	45	0	0	0	0
10	62	62	62	0	0	0	0
11	58	58	58	0	0	0	0
12	88	88	88	0	0	0	0
13	230	190	190	0	0	0	0
14	342	208	178	0	30	0	0
15	6428	665	135	44	374	47	65

10.2 Analysis of exceptions and changes made

Looking at Tables 15 and 16 together, the following deductions seem reasonable.

(i) The definition of class 5 is faulty. As was pointed out in Section 8, all the positions in this class are intended to be drawn, but in 15 cases such positions are chosen by the program and on ten occasions, these are <u>not</u> exceptions. However, it is known that all of the stored file moves lead to wins for White. It is therefore necessary to change the definition of class 5 to exclude the 15 positions currently "selected" from membership, whilst ensuring that in 10 cases the same move is chosen by the program, although for a different reason.

(ii) There are 49 exceptions for which CLASSP is 8, for 45 of which CLASSF is also 8.

Either the functions associated with class 8 need to be altered, or some positions need to be excluded from it altogether (it is unlikely that the definition ot the class is faulty since members of the class have been correctly chosen 223-49 = 174 times).

(iii) There are 135 exceptions tor which CLASSP is 15, for 118 of which CLASSF is also 15.

However, the large number of correctly selected positions corresponding to class 15 and their distribution throughout the various selection levels suggests that it is not likely to be profitable to change the associated functions for that class, but rather to split the class into two or more smaller classes each with its own class value and associated functions.

These appear to be the major causes of the effects shown in the two tables, with other small problems concerned with class 13 and 14 superimposed.

A decision therefore needs to be made whether to begin by looking at class 5, 8 or 15.

As a general rule, when there are many exceptions to deal with, it would seem to be a good idea to concentrate on the class which is evaluated earliest (5 rather than 8 or 15 in this case), since making changes to the algorithm at this level may cause some of the later exceptions to disappear altogether (by affecting the number of

positions belonging to each of the later classes).

However, as there were so few exceptions involving class 5, it was decided to begin by looking at the exceptions associated with class 8.

10.2.1 Problems with the Rook Pawn

Looking at the 49 positions with CLASSP = 8 reveals one significant fact. In every one of these cases the Pawn is on the Rook file (WP1=1) in the initial position (and since the Pawn's file is a static feature of this endgame, this will also be true for all possible successors of those positions). Most chess textbooks (e.g. Fine (1941)) suggest that positions with a Rook Pawn may be a "special case". (As stated in Section 8, this consideration was ignored when setting up the original form of the algorithm in the belief that the general rules embodied there catered for such positions also.)

Thus it would seem reasonable to isolate all initial positions with a Rook Pawn into a second test file and analyse these independently of the original file, with the aim of isolating possible new classes which relate to that type of position only. Of the 193 exceptions listed previously, 54 correspond to initial positions with a Rook Pawn, out of 148 such positions altogether (37%). Recomputing the latter two tables above for Rook Pawn positions only gives the following breakdown, firstly for exceptions only:

Table 17 Summary of exceptions: initial algorithm, Rook Pawn only

CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
5	5	1		1		
8	5	2				
8	8	45	38	7		
8	14	2				
15	5	4				

and secondly for all positions:

Table 18 Selection table: initial algorithm, Rook Pawn only

CLASS	TOTAL	SELECTED	Selection Level 1	2	3	4	TIES
1	98	0	0	0	0	0	0
4	34	34	34	0	0	0	0
5	215	2	0	0	1	0	1
8	302	84	22	40	7	15	0
11	4	4	4	0	0	0	0
13	13	0	0	0	0	0	0
14	10	8	8	0	0	0	0
15	210	16	6	3	7	0	0

At this stage, the principal focus of attention was on the Rook Pawn positions where CLASSP = 8. Inspecting some of the positions where CLASSP = CLASSF = 8 suggested that many of the positions played to by the program should not be members of Class 8 at all. They were poor positions which should be avoided by White.

Thus an attempt was made to specify one or more new equivalence classes to which these 'poor' successor positions could be allocated.

To be effective any such new classes should appear before Class 8 in the testing order and have a lower class value, so that the poor successor positions will be allocated to these classes rather than class 8 and be ranked below the position given by MOVEF (which will still belong to class 8). One rule, given in some textbooks in the case of a Rook Pawn (e.g. Fine (1941),is that if the Black King can reach square C8 then the game is a draw (excluding situations where the Pawn can immediately move to the eighth rank, etc.).

To incorporate this rule into the algorithm a further equivalence class, class 16, was introduced with the definition

Rook Pawn AND dist (BK1,BK2,3,8)<dist (WK1,WK2,3,8).

(The second part of this definition is simply that the Black King is closer to C8 then the White.)

Note that for a Pawn on the H-file, the square C8 would need to be replaced by F8. For the positions under consideration, however, it can be assumed that a Rook Pawn is always on the A-file and that the file two away from the Pawn is therefore always the C-file. No associated functions were defined for Class 16, since the positions belonging to it were intended to be rejected and not accepted. It was decided to test rule 16 between the existing rules 5 and 6, both of which are also intended to correspond to draws, with a class value between those of these two classes. With this change the two tables were recomputed for all the initial positions with a Rook Pawn (since only these could be affected), and it was found that 16

positions which had previously produced "exceptions" were now correctly handled by the program (reference numbers 10,11,13,14,15, 16,60,61,63,64,65,66,107,108,110 and 113*). In all cases these were positions where previously CLASSP = CLASSF = 8, with a selection level of 2.

No position which was previously correctly treated now produced an exception, but some of those which still failed, now failed with a different "program move" (but still one for which CLASSP = 8). This was, in fact, a common feature in the subsequent experimentation. Clearly this change was successful since it corrected 30% of the Rook Pawn exceptions (8% of the overall total) without introducing any new exceptions and it was therefore accepted. This kind of change - introducing a new low-ranked equivalence class - was the most common type of modification which occurred during this experimentation. Such a change can be used effectively to eliminate certain program moves from consideration, by specifying a low-ranked class to which all the corresponding successor positions will then belong. This class must be tested for before the class or classes of which the positions arising after the file moves are members and must have a lower class value than any of these. It is particularly appropriate when CLASSP and CLASSF are equal and a large number of correctly selected program moves with the same value of CLASSP suggests that the class itself is generally well-chosen. It is assumed here that the definition of an equivalence class should be a <u>generalization</u> of the characteristics of the positions which it is required should belong to that class. Thus it would not be satisfactory to take, say, the 45 cases previously referred to where CLASSF = CLASSP = 8 and define a class containing all 45 of the positions after the file moves <u>and no others</u>. It is extremely unlikely that an algorithm

---

* These numbers refer to the 148 positions with a Rook Pawn <u>only</u>. They can be related to the reference numbers in Table 14 by adding 1585 in each case.

developed in this way would perform well for positions other than those in the test file under investigation. Nevertheless, there may be some advantage in dealing with a small number of "special cases" in this way, as will be described later.

From the form of the algorithm described above, with now 54-16 = 38 Rook Pawn positions giving exceptions, the next stage was a further analysis of the remaining 29 positions for which CLASSP and CLASSF are both 8. One feature which appeared to be common to many of these was that the program move allowed Black to move into 'horizontal opposition' (i.e. two files away from the White King on the same rank) with the White King on the Rook file, in front of the Pawn, in each case. Considering Black's best reply is often an important requirement when deciding why a program move is inferior to the one held on file.

Again, a low-ranked equivalence class, class 17, was defined to include positions which need to be avoided, namely all those where Black can take the horizontal opposition with the White King in front of the Pawn. In this case, these can be specified by the predicate function

Rook Pawn AND WK1=WP1 AND WK2>WP2 AND dist (BK1,BK2,3,WK2)=1.

As before, it was decided to test rule 17 between rules 5 and 6, with a class value between those of these two classes and no associated functions.

With this change, the number of exceptions (in the case of a Rook Pawn) was reduced from 38 to 20 (positions 18,19,20,21,22,24,26,28,30, 68,71,73,75,78,114,116,119 and 138* now being correctly handled and no new exceptions created). Once again the change was evidently

* Again these numbers can be related to those in Table 14 by adding 1585 in each case.

successful and was accepted.

Two further changes were now attempted, both of which were unsuccessful and were reversed.

The first was to introduce an extra class, which was tested for immediately after 17. This was defined by

Rook Pawn AND WK1≠WP1

with class value slightly better than the residual class 15, and the same associated functions as class 8.

The second change was to assign all Rook Pawn positions which did not belong to class 17 (or any of those tested before 17) to the residual class, 15. Both these changes resulted in a large number of new exceptions and were rejected.

With the number of instances of CLASSP = CLASSF = 8 now reduced to only 11, from 45 originally, it was decided to re-examine the full test file of 1733 positions, rather than the Rook Pawn cases alone.

10.2.2 Problems with a Pawn on the second rank

The breakdown of exceptions for the full file was now as follows:

Table 19 Summary of exceptions: revised algorithm I

CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
5	5	5	0	5	0	0
8	5	2				
8	8	11 (45)	7 (38)	0	4 (7)	0
8	14	2				
13	15	4				
15	5	17				
15	15	118	4	49	26	39

(previous figures are in parentheses, where different)

Attention was now focused on the 24 cases in which CLASSF = 5, although class 5 was intended to contain only positions which are drawn (with the Black King closer to the Pawn than the White King). A common feature of all these positions was that the Pawn was on the second rank and this suggests that the precise way in which the distances of the Kings from the Pawn need to be compared may be different when the Pawn is on the second rank (and thus has the option of moving either one or two squares initially). A number of attempts were now made to respecify the definition of rule 5. During the course of this, many of the previous exceptions became corrected, and at various stages a number of positions which had previously been correctly handled became exceptions, the definition being refined each time to deal with these. Eventually the definition was changed so that the following additional condition had to be satisfied whenever the pawn was on the second rank

$$\underline{\text{dist}}\ (BK1,BK2,WP1,WP2+1) < \underline{\text{dist}}\ (WK1,WK2,WP1,WP2+1),$$

i.e. the Black King must also be closer to the square in front of the Pawn than the White.

With this change, CLASSF was no longer 5 in any of the 24 cases under consideration: 22 of these positions were now correctly handled, however two positions (1591 and 1612) remained as exceptions but with CLASSF no longer 5.

The full breakdown of exceptions was now the following:

Table 20 Summary of exceptions : revised algorithm II

CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
8	8	12 (11)	8(7)	0	4	0
8	14	3 (2)				
13	15	4				
15	15	118	4	49	26	39

(The increase in the totals for the cases CLASSP = CLASSF = 8 and CLASSP = 8, CLASSF = 14 are accounted for by positions 1612 and 1591, respectively.)

10.2.3 Problems with the residual class

Attention was now focussed on class 15. The final row of the table above shows the selection level for the 118 cases of exceptions where CLASSP=CLASSF=15. The complete table of selection levels for class 15 for all the 538 positions where a choice was made between two successor positions in class 15 was now:

Table 21 · Selection Table: revised algorithm II, class 15 only

Total	Selection Level			
	2	3	4	Ties
538	52	374	47	65

As stated previously, the most appropriate course of action now seemed to be to split up class 15 into two or more smaller classes, each with its own associated functions.

However, it was decided to investigate one further problem before taking this action. From Tables 20 and 21 it can be seen that there are 26 cases in which the correct move was played on the basis of an arbitrary resolution of a tie. Since the test file contained only positions where White has only one move to win, this situation is not satisfactory and a tentative attempt was made at changing the associated functions for the residual class to solve this problem. Based on an analysis of some of the exceptions for which CLASSP=CLASSF=15, a further associated function (8) was defined with value 8 - abs (WK2-WP2) and the associated functions for CLASS 15 were changed to 4,1 and 8 in that order. With this choice, the smallest value of the difference between the Pawn's file and the White King's file would be chosen and subject to this the largest value of the Pawn's rank and the smallest difference in ranks between the White King and the Pawn. Recomputing the table of exceptions with this change showed that there were now 174 exceptions against only 137 previously. 39 of the previous exceptions were now corrected, but there were 67 new exceptions, which were previously correctly handled. From this result it appeared that whilst the original choice of functions 2,3,7 was appropriate in some situations, the

new choice of 4, 1, 8 was preferable in others. It therefore seemed reasonable to split up class 15 into two new classes, one with each of these sets of associated functions. An examination of the 67 positions which had become exceptions despite being correctly handled previously, showed that in every case after White's move had been played, his King was on the Pawn's file, below the Pawn. This suggested that associated functions 4, 1, 8 should only be applied to positions where White's King is not below the rank of the Pawn (function 4 is used to choose the smallest distance between the King's file and the Pawn's file, but this was not intended to lead to the King moving behind the Pawn). On the basis of the above considerations, the algorithm was changed as follows: the original associated functions (2,3,7) for the residual class 15, were restored and a new class 18, was introduced with definition

$$WK2 \geq WP2.$$

This class was placed immediately before class 15 in order of testing, with a value between that of class 15 and that of the next highest valued class, 14. With this change the 67 'new' exceptions disappeared again, leaving a total of 107 remaining. Successful though these changes appeared, an inspection of the full selection table (not given here) showed that the original problem of the 26 positions correctly handled only on the basis of resolving a tie was unaltered, except that the class concerned was now 18 not 15.

The exception table is given below. One important difference from the previous table will be seen to be that where class 15 had previously appeared, class 18 now appeared in its place. The figures in parentheses take this change into account.

Table 22 Summary of exceptions : revised algorithm III

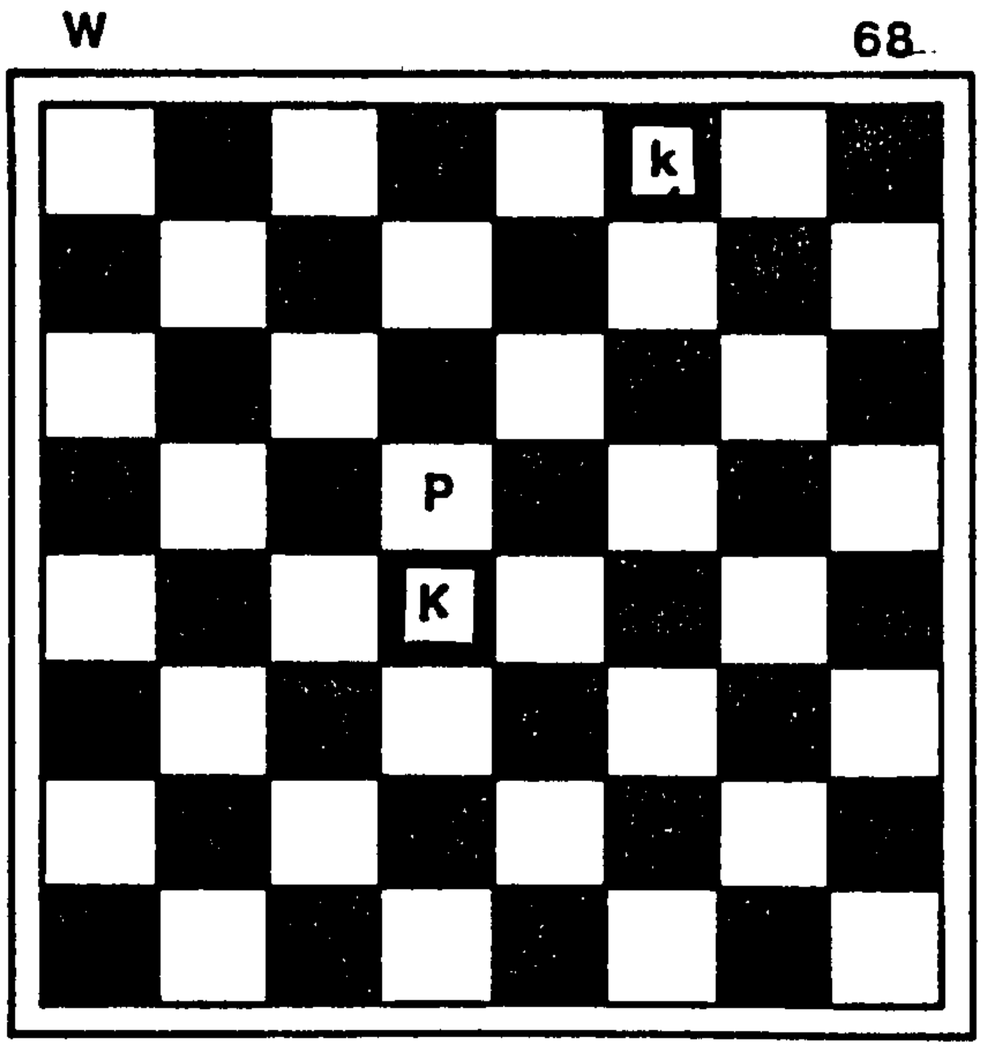
CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
8	8	12	8	0	4	0
8	14	3				
13	18	4				
18	18	83 (118)	0	0	44(26)	39
18	15	5				

An analysis of the 83 positions for which CLASSP=CLASSF = 18 now showed that in many cases the program move was to the same rank as the Pawn, whereas the file move was to the rank above. A change was therefore made to the definition of class 18 to be WK2>WP2, i.e. the case of equality was removed from the previous definition. After this change only 63 exceptions still remained.

Table 23 Summary of exceptions : revised algorithm IV

CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
8	8	12	8	0	4	0
8	14	3				
13	15	4				
15	15	44 (83)	0	5(0)	0(44)	39

It will be seen that the majority of exceptions were now once again where CLASSP=CLASSF=15. The problem of the 26 ties still remained, except that the class involved was again 15. Of the 44 positions where CLASSP=CLASSF=15, 39 were now of the following kind:



Here the only move to win (the file move) is K-C5, moving to the side of the Pawn the more distant from the Black King. A likely continuation would be 1.K-C5, K-E7; 2.K-C6,K-D8; 3.K-D6, taking the opposition and winning.

The program move, however, is K-E5 which leads only to a draw, e.g. 1.K-E5, K-E7; 2.P-D6ch, K-D7; 3.K-D5, K-D8; 4.K-E6, K-E8; 5.P-D7ch, K-D8; 6.K-D6 stalemate. Both these moves are equally valued by the algorithm with K-E5 chosen arbitrarily. For these positions to be handled correctly, it seems to be necessary either to introduce a new equivalence class, as for many of the previous changes, or to introduce a further associated function for class 15. The latter approach was chosen and a new function (number 9) was defined with value abs (WK1-BK1). Choosing the largest value of this function whenever there was a tie ensures that positions such as Figure 68 would be correctly handled. After this change only 24 exceptions remained.

Table 24 Summary of exceptions : revised algorithm V

CLASSP	CLASSF	Number of exceptions	Selection Level 2	3	4	Ties
8	8	12	8	0	4	0
8	14	3				
13	15	4				
15	15	5(44)	0	5	0	0(39)

An inspection of the full selection table at this stage showed that the 26 positions previously discussed although still correctly chosen, were now selected by means of the associated functions not as the result of a tie.

10.2.4 The remaining exceptions : catering for the special case

A complete table of the remaining exceptions is given below.

Table 25 Exceptions with revised algorithm V

REF	POSITION	MOVE F	MOVE P	CLASS F	CLASS P	SEL
524	141745	2	1	15	13	1
537	252745	9	3	15	15	3
584	656745	9	7	15	15	3
596	747745	8	1	15	13	1
1071	151735	9	3	15	15	3
1116	555735	9	7	15	15	3
1124	646735	8	1	15	13	1
1543	454725	9	7	15	15	3
1552	545725	8	1	15	13	1
1591	143312	2	1	14	8	4
1602	284712	5	10	8	8	2
1608	174512	4	10	8	8	2
1612	174412	4	10	8	8	2
1641	143313	2	1	14	8	1
1652	284713	5	9	8	8	2
1654	164613	3	2	8	8	4

REF	POSITION	MOVE F	MOVE P	CLASS F	CLASS P	SEL
1655	174613	4	3	8	8	4
1659	174513	4	9	8	8	2
1662	174413	4	9	8	8	2
1689	153414	2	1	14	8	1
1696	174714	3	2	8	8	4
1697	184714	4	3	8	8	4
1700	184614	4	9	8	8	2
1703	184514	4	9	8	8	2

The complete selection table is as follows.

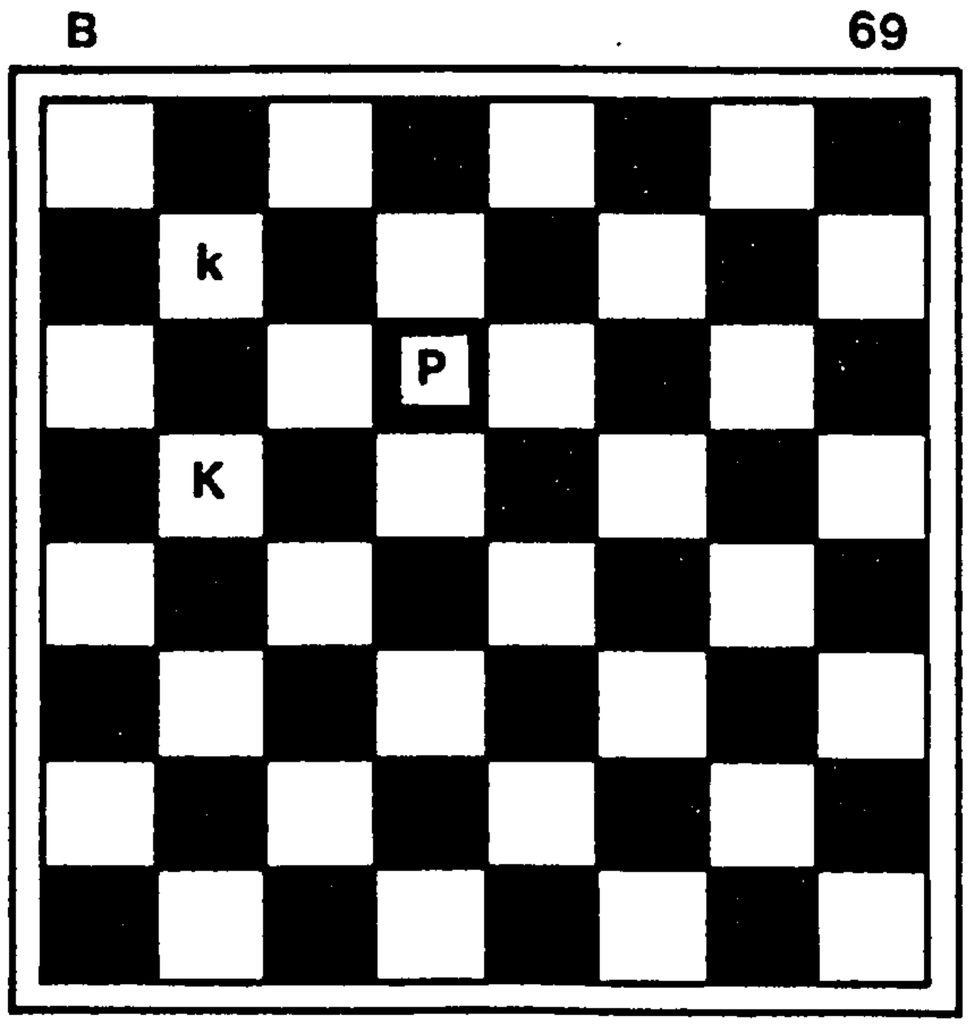
Table 26 Selection Table : Revised algorithm V

CLASS	TOTAL	SELECTED	Selection Level 1	2	3	4	5	Ties
1	1367	0	0	0	0	0	0	0
2	10	0	0	0	0	0	0	0
4	179	179	179	0	0	0	0	0
5	2367	0	0	0	0	0	0	0
6	652	0	0	0	0	0	0	0
7	84	0	0	0	0	0	0	0
8	262	240	226	9	0	5	0	0
9	45	45	45	0	0	0	0	0
10	62	62	62	0	0	0	0	0
11	58	58	58	0	0	0	0	0
12	88	88	88	0	0	0	0	0
13	222	190	190	0	0	0	0	0
14	343	208	178	0	30	0	0	0
15	4437	444	84	25	249	21	65	0
16	255	0	0	0	0	0	0	0
17	82	0	0	0	0	0	0	0
18	1921	219	41	107	0	71	0	0

It will be seen that no positions are now selected solely by arbitrary resolution of a tie.

(For convenience a fourth function has been assigned to each class, all of which are null except in the case of class 15. A selection level of 5 indicates that the fourth function was needed to choose MOVEP.)

Attention was now directed towards the five exceptions for which CLASSP=CLASSF=15, all of which have a selection level of 3. On considering the positions after the file moves in each of the five cases, a common pattern was apparent. The positions were all of this kind:



White has just played 1.P-D6 and now wins, for example by 1....K-C8; 2.K-C6, K-D8; 3.P-D7, K-E7; 4.K-C7 etc. This manoeuvre only wins with the Pawn on the sixth rank. With the Pawn on any lower rank Black has the alternative of moving his King back along the Pawn's file on move 3, and thus drawing.

The common features of the five successor positions arising after the file moves were used to define another equivalence class, 19, with the definition

WP2=6 AND WK2=5 AND BK2=7

AND WK1=BK1 AND abs(WK1-WP1)=2

i.e. the two Kings are on the same file, two files away from the Pawn with the White King on the fifth rank, the Black King on the seventh rank and the Pawn on the sixth rank.

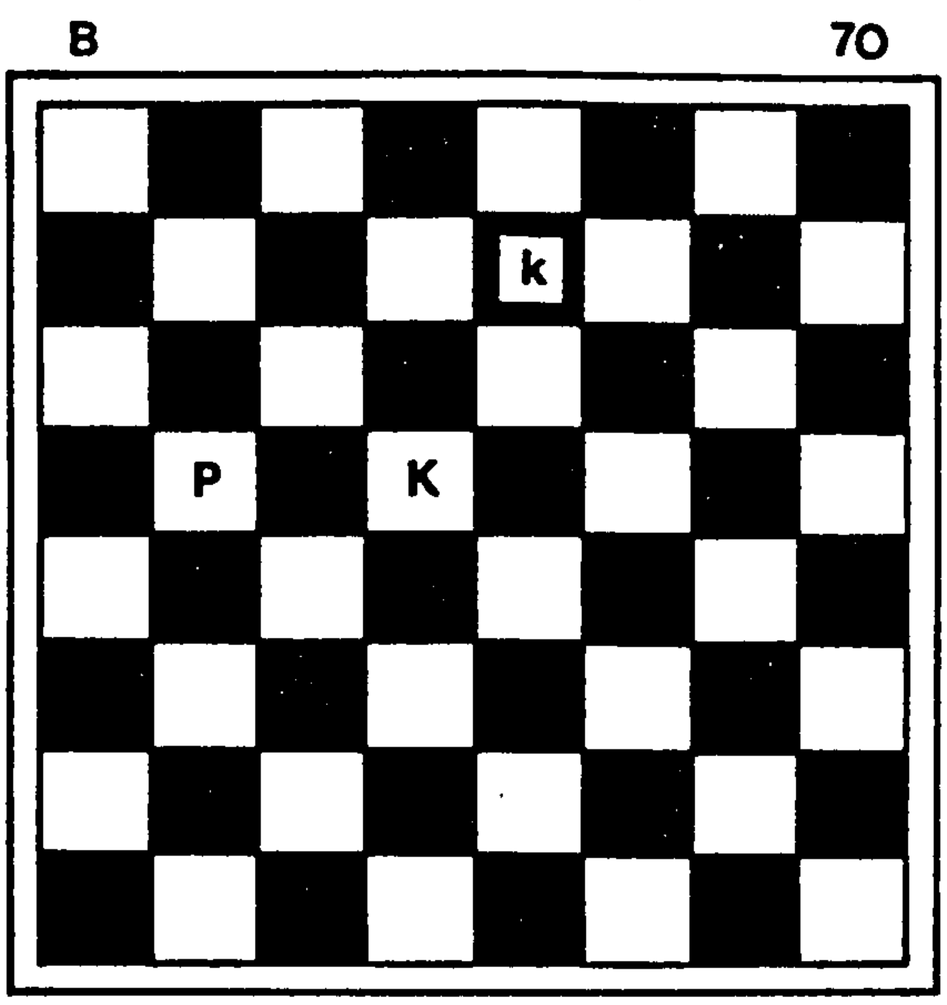
Class 19 was placed immediately above the residual class, 15, in order of testing with a class value between those of classes 18 and 15 (no associated functions). These choices were made to ensure that the file moves would be chosen for the five exceptions under consideration, but that no other positions would be affected. It should be noted that this is unlike the previous changes described, since its object was essentially to upgrade the file moves rather than downgrade the program moves.

This and the following changes to the algorithm were concerned with introducing equivalence classes specifically designed to cater for small numbers of "special cases". Although it was previously stated that such modifications are unlikely, in general, to lead to an algorithm which is correct for positions not on the test file, this consideration does not apply with the same force in the final stages of the experimentation, where the number of positions involved is likely to be small. Given an algorithm which is correct in the great majority of cases tested, it is reasonable to suppose that those remaining are genuine "special cases", the special nature of which is a consequence of the peculiarities of the endgame under

consideration, not of the structure imposed by the equivalence class model or the algorithm.

In the case of Figure 69, for example, if the pieces were all moved down one rank, White would no longer win. The condition WK2=5 AND BK2=7 etc. is a requirement imposed by the stalemate rule and the effect of the edge of the board on the outcome of the game. The problem of the possible proliferation of equivalence classes, each dealing with a small number of special cases, will be taken up later. An examination of the four exceptions for which CLASSP=13 and CLASSF=15 showed that all the positions occurring after the file moves were similar to Figure 70.

B 70



This position is a win for White since Black must prevent White from playing to C6 by 1....K-D7. However, White can now win by playing 2.P-B6 giving a position in class 19. Just as in the play from Figure 69, White only wins because the Pawn is on the sixth rank, so here the transposition to a member of class 19 depends on the

Pawn being on the fifth rank initially. To deal with these positions an additional equivalence class, 20, was defined to include the positions after the file moves in each case. The rule defining the class was as follows

WK2=WP2=5 AND BK2=7 AND abs (WK1-WP1)=2 AND

abs (BK1-WP1)=3 AND sameside

where sameside is true if both Kings are on the same side of the Pawn. Thus the rule specifies that the White King and the Pawn are both on the fifth rank with the Black King on the seventh rank, with the White and Black Kings respectively 2 and 3 files from the Pawn on the same side.

The class was placed immediately above class 15 in order of testing, i.e. below class 19 (although the order of these two classes is irrelevant). A class value was chosen so that class 20 would rank above class 13, but below the next highest ranked class, since CLASSP=13. No associated functions were defined.

The remaining 15 exceptions all involved Rook Pawns and in each case CLASSP=8. It might therefore be expected that it would once again be necessary to define one or more low-valued new classes of "positions to be avoided".

In fact, four such classes were defined, all placed for convenience between classes 16 and 6 in order of testing and with low class values (note that the classes must come before Class 8 in the testing order or CLASSP will still be 8). The classes were defined as follows

CLASS 21 Rook Pawn AND WK1=WP1 AND WK2≥WP2+2 AND dist(BK1,BK2,3,WK2-1)=1.

CLASS 22 Rook Pawn AND WP2>2 AND WK2=WP2+4 AND dist(BK1,BK2,3,WP2+2)=1.

CLASS 23 Rook Pawn AND WK1=WP1 AND WK2=WP2+1 AND dist(BK1,BK2,3,WK2+1)=1.

CLASS 24 Rook Pawn AND WK1=WP1 AND WK2=WP2+3 AND dist(BK1,BK2,3,WK2-2)=1.

In each case no associated functions were defined and class values between those of classes 5 and 6 were chosen. It was expected that these classes would never be selected as best and so their relative class values and ordering were not significant.

Classes 21-24 are all examples of special cases where the Pawn is on the Rook file and Black to move can play to a suitable square on the Bishop file and thus draw. They are therefore all positions to be avoided by White.

10.2.5 Conclusions

With these new classes included, the program worked correctly for all 1733 positions on the test file. An analysis of the number of successor positions belonging to each equivalence class and those chosen at each selection level is given in the table below, with the classes arranged in testing order.

For each equivalence class, the class value is also given.

Where the relative order of testing of two classes has not previously been specified, the order given here is arbitrary. The class values for the new classes have been chosen to lie in the ranges previously specified but are otherwise also arbitrary.

Table 27 Selection Table : final algorithm

CLASS	TOTAL	SELECTED	Selection Level 1	2	3	4	5	TIES	Class Value
1	1367	0	0	0	0	0	0	0	10
2	10	0	0	0	0	0	0	0	20
4	179	179	179	0	0	0	0	0	140
5	2367	0	0	0	0	0	0	0	30
16	255	0	0	0	0	0	0	0	34
17	82	0	0	0	0	0	0	0	32
21	12	0	0	0	0	0	0	0	31
22	7	0	0	0	0	0	0	0	33
23	1	0	0	0	0	0	0	0	35
24	3	0	0	0	0	0	0	0	36
6	652	0	0	0	0	0	0	0	40
7	84	0	0	0	0	0	0	0	50
8	239	237	237	0	0	0	0	0	130
9	45	45	45	0	0	0	0	0	120
10	62	62	62	0	0	0	0	0	110
11	58	58	58	0	0	0	0	0	100
12	88	88	88	0	0	0	0	0	90
13	222	186	186	0	0	0	0	0	80
14	343	211	181	0	30	0	0	0	70
18	1921	219	41	107	0	71	0	0	65
19	13	13	13	0	0	0	0	0	64
20	4	4	4	0	0	0	0	0	85
15	4420	431	84	25	236	21	65	0	60

As for Table 7 in Section 7, the thick line is used to indicate the number of functions associated with each class, two for class 1, none for class 2 etc.

A number of conclusions can be drawn from a careful consideration of this table.

The presence of zeros within the thick lines, for selection levels 2,3 and 4, shows that several of the associated functions have played no significant part in the move-finding process. No positions in classes 1,5 or 6 have been selected by the program, and although many positions in classes 4,8,9,10,11 and 13 have been selected, in each case the selection was made at level 1, i.e. by class value alone. For classes 14 and 18, the situation is more complicated.

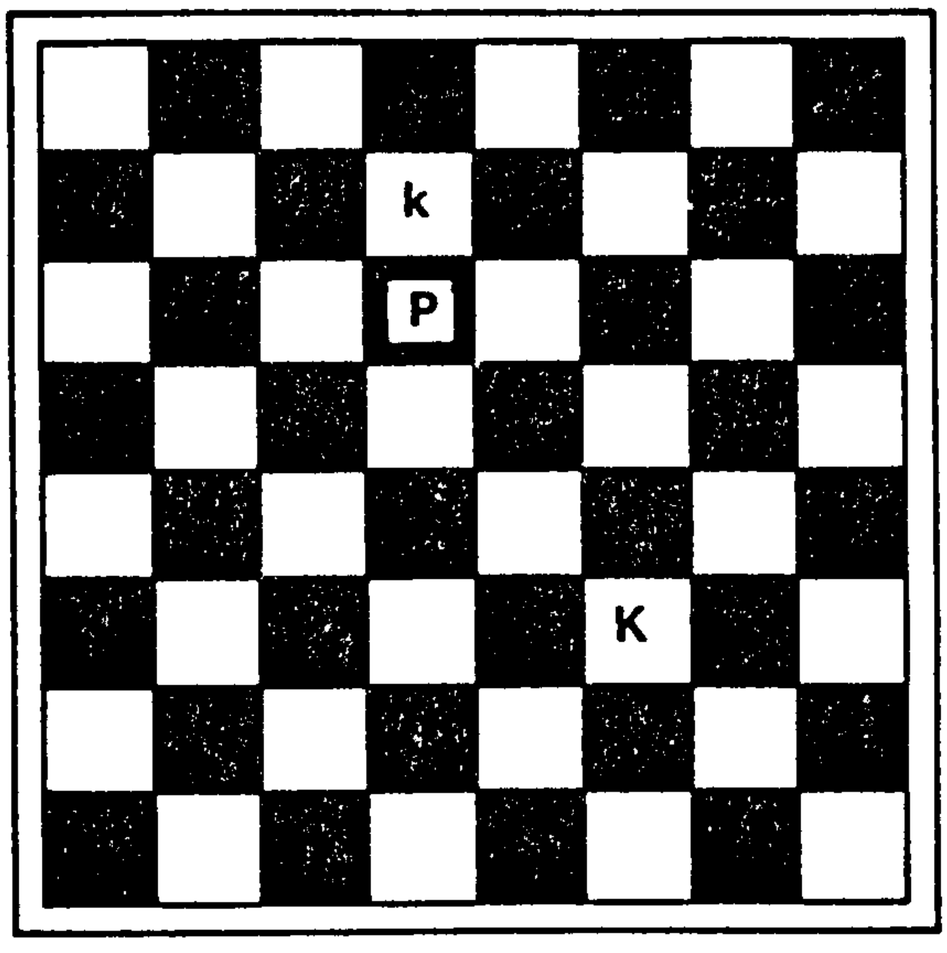
In 30 cases it has been necessary to make a choice between two or more positions in class 14, but in no case was it necessary to use the third associated function to make the choice. The zero at selection level 2 indicates that in each of the 30 cases, the first associated function was insufficient on its own to resolve the tie. This does, in fact, provide some "negative" evidence in support of the use of this function, since a poorly chosen function is unlikely to prove innocuous.

Similar considerations apply for the zero at selection level 3 for class 18. All the associated functions corresponding to zeros in the table could be removed from the algorithm altogether without affecting the results obtained for the test file under consideration. It must be borne in mind, however, that this file contains a (non-random) sample of all possible positions for the King and Pawn against King endgame. It would therefore be advisable to analyse a larger sample of positions before making a firm decision.

Considerations strictly unrelated to the choice of a correct algorithm may be relevant here. For example, in Figure 71, whatever move White makes, the position belongs to equivalence class 1,

W

71

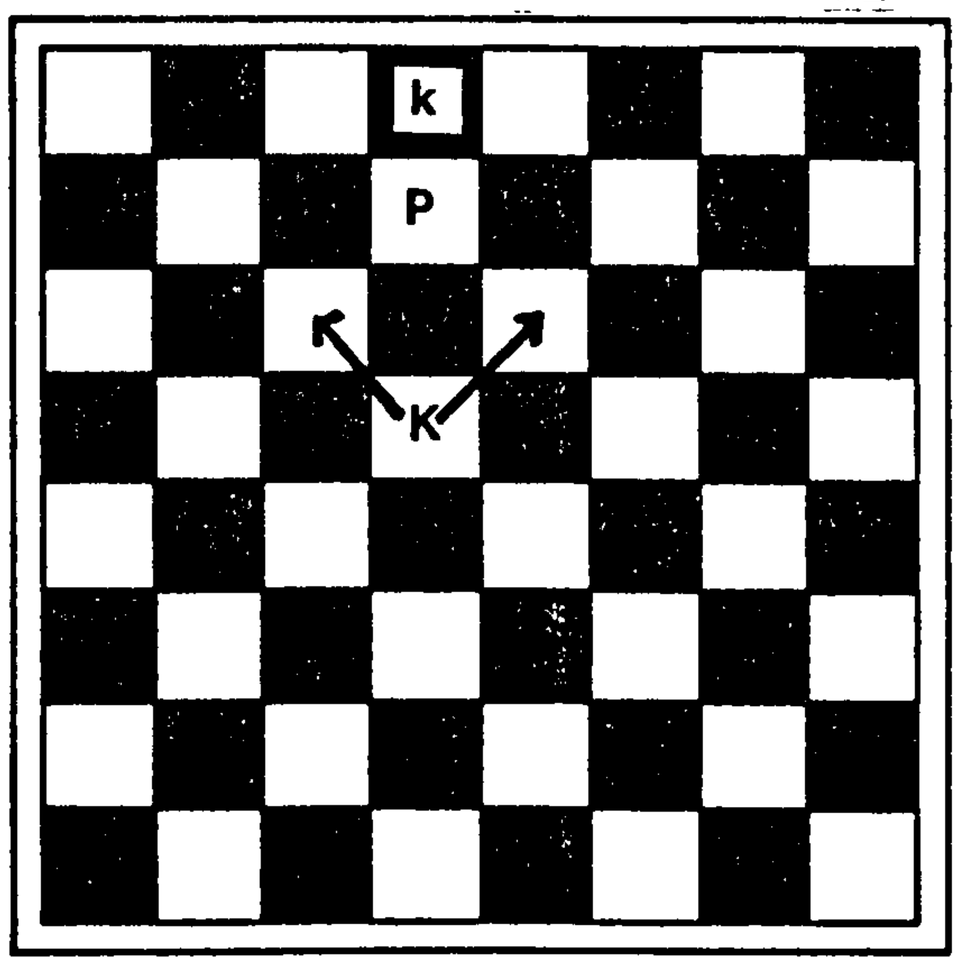


that is Black can immediately capture the Pawn and the game is drawn. Although White's move is strictly irrelevant, the associated functions for class 1 will ensure that the most sensible looking move K-E4 is chosen. In this respect, it would seem reasonable to retain the associated functions for that class although they are strictly unnecessary.

On the positive side, the choices of the second function for class 14, the first and third functions for class 18 and all four functions for class 15 are well supported. Thus there are 65 "confirming instances" of the correctness of the fourth function for class 15 and 347 (=25+236+21+65) for the first function for that class, etc. There are no cases of a position being chosen arbitrarily to resolve a tie and this is significant, because the test file was chosen so that in each position White had only <u>one</u> move to win.

For other King and Pawn against King positions, however, it is possible for a tie to occur 'legitimately', for example in Figure 72, where White has two winning moves K-C6 and K-E6. Any attempt to choose between these is entirely artificial.

W 72



Although the number of equivalence classes used by the algorithm has risen from the original 15 to 24, this number could be substantially reduced using the fact that if any set of classes occur consecutively in testing order and have class values in a consecutive range (in any order), then provided no members of those classes are ever selected, they can be combined into one class. The defining rule for this new class is just the logical 'OR' of the rules for each of the component classes and any one of the class values can be taken for the new class.

Thus, for the positions on the test file it is possible to combine classes 1 and 2 into a single class and also classes 5,16,17,21,22, 23,24, 6 and 7. The class values of this latter group (30,34,32,31, 33,35,36,40 and 50) form a "consecutive range" for this purpose, since there is no class with a value of 45, for example. By making both of these combinations the number of classes would be reduced again to 15.

It must be borne in mind, however, that combining classes in this way does not, in general, reduce the complexity of testing required. In the case of classes 1 and 2, the ideas involved (Pawn en prise and stalemate) are logically distinct and there is no advantage in combining them except to reduce the overt size of the algorithm.

Where there is one or more common predicate, however, combining the class definitions enables a simplification to be made. Thus classes 16,17,21,22,23 and 24 are all related by having a common predicate 'Rookpawn' and combining them enables this condition to be tested only once. Looking for such combinations would probably be even more valuable in the case of more complex endgames.

In such cases, where classes share a common property, it is also more likely that the combined classes will still be valid for positions outside the test file. Nevertheless, without further analysis the correctness of such combinations cannot be conclusively demonstrated.

A separate "robustness analysis" of the final algorithm reveals that deleting class 11 altogether would produce no exceptions for the test file. It would therefore be possible to reduce the final form of the algorithm to only 14 classes if required. Once again, without

analysing further positions, such a change cannot be made, with absolute certainty that it is correct, for positions other than those tested.

The summary that follows shows the final form of the algorithm, assuming that classes 16,17,21,22,23 and 24 are combined into one new class (which will be numbered 16), with a class value of 31, and that no other changes are made. There are 19 equivalence classes in this form of the algorithm. It would, of course, be possible to renumber the classes themselves from 1 to 19 (in order of testing) and their values from 1 to 19 also (in ascending order of favourability, from White's viewpoint). For clarity, no such renumbering has been carried out here. For completeness, class 3 (Pawn on eighth rank) has again been included in the summary.

Even after substantial development and testing there may still be a certain degree of arbitrariness in the algorithm as given here, particularly in the testing order of certain classes. Once again further testing is necessary before a final decision concerning the appropriateness of all the possible changes can be made. Classes 19 and 20 are examples of "special cases", classes with only a small number of members. In both cases, however, the class has been included to deal with a small number of difficult situations of an exceptional nature. They may be said, therefore, to have been originated in response to the natural demands of the endgame itself, and in that sense they play a small but important part in the final algorithm.

10.3 Summary of the final algorithm

The value table for the final King and Pawn against King algorithm with 19 equivalence classes (i.e. after combining the previous classes 16,17,21,22,23 and 24) is given below

Table 28 The final algorithm (19 equivalence classes)

Class (in order of testing)	Value	Associated functions
1	10	2,3,0,0
2	20	0,0,0,0
3	150	0,0,0,0
4	140	1,0,0,0
5	30	2,3,0,0
16	31	0,0,0,0
6	40	1,2,3,0
7	50	0,0,0,0
8	130	1,4,7,0
9	120	1,0,0,0
10	110	1,0,0,0
11	100	1,0,0,0
12	90	0,0,0,0
13	80	1,0,0,0
14	70	1,6,5,0
18	65	4,1,8,0
19	64	0,0,0,0
20	85	0,0,0,0
15	60	2,3,7,9

The selection table for this form of the algorithm is the same as Table 27 in Section 10.2.5, with Classes 16,17,21,22,23 and 24 combined.

The differences between the final 19 equivalence classes and those given in the initial form of the algorithm in Section 8 are as follows.

New Classes

(i) Class 16, defined by:

Rook Pawn AND {[WK1 = WP1 AND WK2 > WP2 AND dist (BK1,BK2,3,WK2) = 1]

OR [dist (BK1,BK2,3,8) < dist (WK1,WK2,3,8)]

OR [WK1 = WP1 AND WK2 ≥ WP2+2 AND dist (BK1,BK2,3,WK2-1) = 1]

OR [WP2 > 2 AND WK2 = WP2+4 AND dist (BK1,BK2,3,WP2+2) = 1]

OR [WK1 = WP1 AND WK2 = WP2+1 AND dist (BK1,BK2,3,WK2+1) = 1]

OR [WK1 = WP1 AND WK2 = WP2+3 AND dist (BK1,BK2,3,WK2-2) = 1]}

(No associated functions.)

(ii) Class 18, defined by:

WK2 > WP2

(Associated functions 4,1,8,0 in that order.)

(iii) Class 19, defined by:

WP2 = 6 AND WK2 = 5 AND BK2 = 7 AND WK1 = BK1 AND abs(WK1-WP1) = 2

(No associated functions.)

(iv) Class 20, defined by:

WK2 = WP2 = 5 AND BK2 = 7 AND abs(WK1-WP1) = 2

AND abs(BK1-WP1) = 3 AND sameside

(where 'sameside' is true if both Kings are on the same side of the Pawn).

Changes to existing classes

(i) Class 5: the additional condition

dist(BK1,BK2,WP1,WP2+1) < dist(WK1,WK2,WP1,WP2+1)

was added in the case where WP2 = 2.

(ii) A fourth associated function (9) was added to class 15.

Associated functions

Two new associated functions were defined: function 8 with value

8 - abs(WK2 - WP2)

and function 9 with value

abs(WK1 - BK1).

Functions 1-7 were unchanged.

10.4 Discussion

In this section the iterative refinement of an initial algorithm for King and Pawn against King has been described in detail.

In this case, refining the algorithm to play perfectly in all the positions in the test file under consideration proved possible with only a relatively small amount of iteration. Moreover, the process involved was one in which feedback from the computer system itself played a significant part. In Section 11, the process is considered from the viewpoint of the facilities which would be required in a semi-automated system of program improvement, or an eventual fully automated system.

The initial algorithm took as its basis the knowledge contained in typical chess textbooks and the formulation of the algorithm consisted of a relatively straightforward translation of this information.

Since one of the criteria set out in Section 1 was that the algorithm should correspond fairly closely to the Chessplayer's understanding of the important features of the endgame, it is appropriate to consider the revised algorithm from this viewpoint.

Although textbooks give general information in the case of a Rook Pawn, in particular that Black draws if he can reach square C8, the implications of this are not worked out in detail. Class 16 contains descriptions of positions which White must avoid if he is to prevent Black from reaching C8 without allowing him to capture the Pawn. For example, White should not play his King on to the Pawn's file, three ranks ahead of the Pawn, if Black can then occupy the C file, one rank ahead of the Pawn. This result can be verified by inspection but is certainly not specifically given in textbooks.

The six original rules which make up the final rule 16 provide an example of iterative refinement in the case of the Rook Pawn. Only the second component

Rook Pawn AND dist (BK1,BK2,3,8) < dist (WK1,WK2,3,8)

is directly derivable from textbook descriptions. Given further experimentation it may be that additional OR conditions would be added to rule 16 and that these could eventually be synthesised to form a new principle for the endgame.

The original form of rule 5 embodies a rule not explicitly given in textbooks, but intuitively obvious: if the Black King is closer to the Pawn than the White King, the result is a draw. The original form of this rule includes one refinement: if Black is diagonally ahead of the Pawn, the count of his distance from the Pawn must be increased by one. As the analysis in Section 10.2.2 shows, a further refinement is necessary in the case of a Pawn on the second rank. Here the Black King must also be closer to the square in front of the Pawn than the White King (but no addition is made if it is diagonally ahead of that square). Once again this is an extension of the information given in the textbook.

In the case of the fourth associated function introduced for Class 15, it has been possible to introduce an important new principle with very little change to the program: "all other considerations being equal, move the White King as many files away from the Black as possible". This modification not only enables White to play correctly in Figure 68, Section 10.2.3, but also in 38 other positions in the test file without in any way designating them as "special cases". These include the position WK on D1, BK on F8, WP on C3 which Clarke (1975) describes as causing all but the best players some trouble. The principle embodied in the use of the fourth associated function is not given in textbooks.

In these cases and those of the other modifications made, the original algorithm has been refined, not rejected. The modifications can be expressed in terms familiar to chessplayers and could readily be incorporated as additions to existing textbooks.

Thus it appears that the process of refinement did not interfere with the property of the original algorithm that it could be closely related to the chessplayer's knowledge of the game; rather it can be seen as

extending that knowledge.

When deciding whether or not to combine two or more equivalence classes, as in Section 10.2.5, consideration should be given to the "naturalness" of the resulting class, as seen by the chessplayer. For this reason, it is probably only worthwhile to combine classes which have a significant common feature, such as a Rook Pawn, Pawn on the second rank, etc.

Since it appears to have been possible to extend the information given in textbooks by iterative refinement, it may also be possible to begin with a much less sophisticated algorithm and develop such concepts as the opposition, or the importance of the C8 square with a Rook Pawn, by a process of refinement and synthesis. However, this is likely to be a difficult task, just as it would be for a chessplayer who did not already know these concepts to discover them for himself. Whereas the computer is able to process a large amount of information very rapidly, the human is much better able to perceive relationships. A combination of these two facilities would therefore seem to offer the possibility of significant progress.

Some of the requirements for a partially or fully automated system of program improvement are discussed in the next section.

10.5 Postscript : the optimality of the final algorithm

Producing an algorithm which performs perfectly in all legal positions with White to move (not just those on the test file) was not itself an aim of the experimentation described in the preceding sections. However, it is believed that some information on the level of optimality of the final algorithm will be of general interest.

For this reason a series of tests was arranged to compare the move played by the algorithm in each of the 62,480 positions where White to move has a forced win, with the theoretical best move or moves as derived from Clarke's database.

The outcome of this testing was that in 2601 positions the algorithm produced a non-optimal move for White.

A cursory inspection of the corresponding exceptions revealed a straightforward omission in the definition of equivalence class 4 (White Pawn can "run"), in which the condition that the White King was not somewhere on the file in front of the Pawn had not been included. Although it is unlikely that this omission will result in White failing to win in any instance, it does result in non-optimal play in many positions such as WK : C7, BK : H1, WP : C6. Here any White move will win and all moves are equally good except K - C8. Unfortunately, since the algorithm considers all the possible successor positions to be of equal value as members of class 4 and the first generated move is K - C8, that is the move which is chosen. Inserting a suitable additional test into the definition of class 4 reduced the total number of non-optimal moves to 2139 out of 62480, i.e. the algorithm played optimally in 96.6% of the positions.

Of these 2139, there were 921 cases (43.1%) where the algorithm failed to recognize a successor position from which the Pawn could "run" against any play by Black. As has previously been stated, class 4 was known to be only a partial implementation of the predicate "Pawn can run".

Of the remaining 1218 "exceptions", there were 753 cases (61.8%) where White had a unique best winning move and 465 cases (38.2%) where there was a choice of equally good "best" moves. In 559 cases out of the 1218 (45.9%) White's Pawn was on the D-file.

Although the total number of non-optimal cases may seen surprising in view of the detailed analysis previously described, it is probably best viewed as an indication of the extreme precision required for optimal play in this endgame. In all of the cases inspected the algorithms move still appeared to win and in many cases the previously expressed conclusion was reinforced that textbook descriptions are not sufficient to demonstrate(either explicitly or implicitly) a perfect winning strategy. It would seem likely, in fact, that an optimal strategy would appear extremely "unnatural" to the chessplayer. A controlled study of the chessplayers endgame knowledge would therefore appear to be a necessary precursor of further research into knowledge representation in this field.

Using the methods of program refinement previously demonstrated, the final King and Pawn against King algorithm has been further modified until at the time of writing only 560 positions remain in which a non-optimal move is made in a position where White can win. Thus 99.1% of such positions are now handled optimally. This modified form of the algorithm includes a provably correct implementation of the "Pawn can run" predicate (class 4), as described in the reprint of the technical report "King and Pawn against King : using effective distance" at the end of this thesis.

Of the 560 exceptions, 441 (78.75%) occur in positions where there is a single best move for White and the worst such cases are with the Pawn on the C and D-files (166 and 174 exceptions, respectively). In all cases inspected the algorithm's move appears to be a reasonable alternative to the "optimal" move stored on the database. Modifying the algorithm has resulted in a net increase of two in the number of equivalence classes used. This further refinement exercise tends to reinforce the conclusions reached in Sections 10 and 11 but does not itself demonstrate any new principles and will not therefore be discussed further.

11. Towards a self-improving system

In the previous section a description was given of the changes made to the original King and Pawn against King algorithm as a result of experimentation with a test file of positions and associated best moves.

Despite one or two points at which the program's performance actually worsened, in general the changes appeared to form part of a convergent process of iteration, reasonably easy to perform (for people) and which preserved the property of the original algorithm of being meaningful in terms of descriptions in chess textbooks and the chess-player's own knowledge of the endgame. This conclusion bears out in a more controlled form the results of the experimentation on the King and Rook against King algorithm, discussed in the first part of this thesis.

The procedure followed in Section 10 can be described as partially-automated. The computer was used to provide information in forms which helped to suggest possible improvements to the algorithm and which enabled the effectiveness of such improvements to be assessed. The computer did not, however, itself decide on changes to be made or modify its own algorithm. In this section, following a brief summary of related work, the extent to which the equivalence class model assists in the modification process will be discussed and the possibility will be considered of implementing a fully-automated self-improving system based on this model and the methods described in Section 10.

11.1 Related work

Many learning programs have been written for a variety of games. The most common approach has been to modify the coefficients in an evaluation function or the entries in a table, for example Smith's (1973) program for partnership dominoes and Samuel's programs (1959 and 1967) for checkers. The methods employed are not, however, directly relevant to the present work in which, as will be seen in Sections 11.3 and 11.4 below, the major difficulty lies in constructing a function describing the characteristics common to a set of positions (or a set of more specific functions) in an appropriate manner. This "function-building" problem also occurs in conjunction with automatic theorem proving, studies of concept formation and inductive generalization and automatic program improvement and verification. The most directly related work would seem to be that of Elcock and Murray and of Popplestone (1969) and Winston (1970).

The former work is described in three papers, the first of which (Elcock and Murray (1967)) is concerned with the means by which descriptions of significant board patterns can be automatically generated by a game-playing program (for the game of Go-Moku), using backtrack analysis of its lost games to identify situations from which it sees itself as lost and adding descriptions of these to an ordered list of subgoals. The second paper (Murray and Elcock (1968)) extends the descriptive notation available and the third paper (Elcock (1968)) considers the problems involved in constructing descriptions in a wider context. Attention is focussed throughout on the problem of finding a minimal description of each subgoal, i.e. of combining and thereby simplifying combinations of descriptions where possible.

The form of description used, however, is specific to games similar to Go-Moku and could not easily be extended to more complex situations such as chess positions.

Popplestone's paper describes an experiment in constructing an "induction engine" to take a sample of the input-output table for a function and generate from it a description of the property common to all members of the sample. An example is taken where the method is used to synthesise a partial description of what constitutes a won position at noughts and crosses. Again, it is not clear whether the method employed would cope satisfactorily with the more complex descriptions necessary for more difficult domains, such as chess end-games.

Winston (1972) describes a program embodying a theory of concept learning based on a small number of carefully selected examples (Winston (1970)).

The program is given a set of primitives together with examples of a concept to be learnt, such as "an arch", and produces a corresponding description of the concept. Further examples are then given one at a time by a "teacher", both of arches and of "near misses", i.e. structures which are similar to but slightly different from arches. After each example, the program makes a corresponding modification to its description.

Winston stresses the importance of the teacher in selecting examples aimed at successively refining the program's descriptions. The program includes domain-specific knowledge which enables it to handle the important and commonly occurring case where there is more than one difference between its current description of the concept and the description of a "near-miss". Thus there is a ranking of possible differences in order of their likely significance.

There are a clearly a number of similarities between Winston's approach and the algorithm refinement described in this thesis. In particular, the use of "near-misses" is similar to refinement by means of exception reporting. Examples have been given where specific a priori knowledge, such as that the Rook Pawn is likely to be a "special case", was used in deciding which of the many features distinguishing exceptions from correctly handled positions was most likely to be significant.

The main difference is that in Winston's case the "teacher" has full knowledge of the concept involved and is thus able to direct the program's learning far more closely than would be possible for even the most elementary chess endgames where no simple description of a correct playing strategy is known. It is also likely that a considerably more sophisticated descriptive language would need to be developed in the case of Chess and the availability and power of this language plays a crucial part in applying Winston's method.

Production Systems

The work referred to above is related to the content of the rules defining the equivalence classes (and the class values and associated functions). In contrast, the method described in Section 2.3 of testing the rules in order, with the first one satisfied being used to determine the class to which each position belongs, corresponds fairly closely to the organization of a condition-driven rule-ordered production system. In fact, the entire move selection process can readily be written in the form of such a production system, beginning with the first generated successor position in the database to be examined by the production rules. In this case the production rules include the rules defining membership of the classes and have a priority ordering of testing. The highest priority rule satisfied (i.e. the first in testing order) is taken in every case and the corresponding action is to calculate the position value (using suitable associated functions) and then to copy the next successor position into the database. The cyclic nature of a production system ensures that by this means all successor positions will be examined in turn and when no further positions remain an additional rule can be used to select the move corresponding to the position with highest value, play it, ask for the opponent's reply, etc.

Production systems themselves have been used to model cognitive behaviour, e.g. by Young (1974).

Waterman (1977) describes one of the advantages of this form of program organization as that it offers a parsimonious way of modelling human cognition which, because the rules represent independent components of behaviour, can be modified in an incremental way. In addition, representing a large body of knowledge in rule form makes it easier to explain, justify and analyze the rationale used by the program to make its decisions. Finally, the simplicity of the control structure facilitates automatic program creation, debugging and verification.

These comments are closely related to those previously made about the equivalence class model described in this thesis. To continue the analogy further, the problem of iterative algorithm refinement discussed in the remainder of this section can be considered as a particular form of the problem of incremental updating of a given production system.

If an automatic method were known of updating a given production system, there would be a corresponding automatic method of updating move-selection algorithms based on the equivalence class model. However, since the basic operation of mapping a position to its value corresponds to only one pass through the production rules and could thus be represented by a simpler control structure than a production system, it is likely that other simpler methods may exist for the latter problem.

11.2 The process of algorithm refinement

Given an initial algorithm and a database of stored positions and the associated best moves, the basic procedure followed in Section 10 can be summarised as follows:

(1) Compute exceptions, by comparing the move generated by the algorithm with that held on the database, for each position.

(2) If there are no exceptions, then stop, otherwise go on to step (3).

(3) Make a tentative change to the algorithm.

(4) Recompute the exceptions with this change incorporated.

(5) (Decision stage). Accept the modification and return to step (2), or reject the modification and return to step (3) to make a different change, or (undecided) make a further modification and return to (4).

In most practical cases, only a sample of all possible positions in the endgame under consideration will be stored on the database and the best moves stored will generally be those played by expert players in specimen games. There will therefore usually be some likelihood that the stored moves are not invariably the best available. In some cases there may also be other moves equally good or moves which lead to the same result either more or less quickly. The objective of program improvement is then to obtain the best 'fit' to the positions on the database while bearing in mind that the positions are only a sample (which may or may not be a random one) of all those which could have been chosen.

The two critical stages of the procedure described above are steps (3) and (5), deciding on a tentative modification to the algorithm and evaluating the result of this change.

The equivalence class model provides assistance at both these stages in a number of important ways.

Firstly, the model is a structural (rather than a procedural) representation and one important advantage of such a representation is that it enables the mechanics of algorithm modification to be performed in a relatively simple way, each of the underlying elements of the algorithm being capable of manipulation independently of the others.

Secondly the use of equivalence classes and associated functions enables a convenient summary of exceptions to be made in the form of a compact table (such as Table 15 in Section 10). Each row of this table corresponds to a group of exceptions with a particular value of CLASSP and CLASSF and the positions concerned can be obtained from a full list of exceptions (which can easily be tabulated together with their corresponding values of CLASSP, CLASSF and 'selection level'). A full "selection table" (such as Table 16 in Section 10) can be used to find the number of "confirming instances" of the choice of each equivalence class and associated function and thus to indicate those parts of the algorithm which may need modification.

The experimentation described previously suggests that looking at exceptions split up into groups in this way is an effective means of isolating those exceptions which are genuinely related by some common property, and that the essential independence of the rules defining

the equivalence classes and of the associated functions (in the sense that a change made to one will not affect any of the others) helps when choosing possible modifications to the algorithm.

The order of evaluation of the rules and the order of precedence of the associated functions enable modifications to be made in a controlled manner where the extent of possible changes to the list of exceptions can be known in advance. Thus, for example, inserting a new equivalence class near the end in order of testing can have no effect on the value of positions (with Black to move) which belong to classes earlier in the testing order.

When evaluating a modification to the algorithm, it is again helpful to consider the tables of exceptions, in combination with the full selection tables, as a means of breaking down the changes in the exceptions produced.

Although in the experimentation described in Section 10, the computer system supplied a great deal of valuable information, it took no direct part in the decision - making process. The possibility of developing a fully automated system of algorithm refinement is now considered.

11.3 Developing a self-improving system

The principal requirements for a self-improving system can be summarized as follows:

(1) A control system to perform the basic cycle of operations described in Section 11.2.

(2) A database of stored positions and associated moves, known from some external source.

(3) An algorithm stored in a form suitable for modification.

In the case of the equivalence class model, the algorithm consists of three principal elements:

(a) a set of predicate functions (rules) defining equivalence classes of positions

(b) a set of associated functions

(c) a value table containing the class value and indices representing the associated functions for each class, each rule corresponding to one row of this table.

Further elements may be added to this list to allow for future modifications to the algorithm, in particular a list giving the order of evaluation of the rules, a set of possible associated functions and a pool of "primitives" from which any new predicate functions can be formed in accordance with pre-determined rules. Provided that no new associated functions or primitives are required and that the rules for forming new predicates are adequate, designing a control system to implement a given change to the algorithm presents no theoretical difficulty.

In the case of the King and Pawn against King algorithm, both initial and revised versions, the primitives were the rank and file co-ordinates of each of the three pieces, i.e. WK1, BK2, etc. and with the exception of the residual class, all predicate functions (rules) were of the following form, using Backus-Naur notation:

<predicate>:: = <expression>| <predicate> AND <predicate>|
<predicate> OR <predicate>| NOT <predicate>

where <expression>:: = <item> <relational operator> <item 2>

<item> :: = <primitive>| <function value>

<item 2>:: = <item>| <constant>

relational operator :: = <|>|≤|≥|=|≠

constant :: = 1|2|3|4|5|6|7|8

and <function value> here means the value of a function taking an appropriate number of arguments.

The standard functions abs (one argument), max and min (two arguments) were used, plus the special function dist which gives the "block distance" in King moves between two squares.

The predicates "Rookpawn" and "Sameside" were introduced during the experimentation stage, but these are only shorthand for <expression>s such as WP1 = 1. With a suitable set of primitives, these rules for forming predicate functions would appear to be sufficient for most or all endgames.

All the associated functions were of the form <item> or 8-<item>. In general, all associated functions should be <item>s, together with an option to enable the largest or the smallest value of the function to be used to resolve ties, in each case.

The basic iterative cycle of algorithm refinement described in Section 11.2 can be broken down into the following main components:

(i) select a "group of exceptions" using the tabulated summary of exceptions in conjunction with the full selection table;

(ii) find a common property possessed by most or all of these exceptions;

(iii) decide on and make a corresponding change in the algorithm;

(iv) examine the resulting change in the exceptions produced and decide whether to accept the change made, reject it or (if undecided) try an additional change.

A number of detailed heuristics for these stages were given in Section 10.

Stage (i), selecting a group of exceptions sharing the same values of CLASSP and CLASSF (and possibly the same selection level), is likely to prove reasonably easy to automate since in principle any choice will suffice, although some will probably help the process to converge more rapidly than others. For example, given several large groups of exceptions, the one involving the class or classes evaluated earliest should be examined first, since making changes to the definitions of such classes (or introducing new classes early in the testing order) may cause some of the other exceptions (involving later evaluated classes) to disappear altogether.

The existence of a priori knowledge may also be valuable here. For example, in Section 10 it was known that no positions should be selected either by arbitrary resolution of a tie or with CLASSP=5. This knowledge proved valuable in pointing to significant groups of exceptions and in revealing that in some cases moves were being correctly selected for the wrong reasons. Such prior knowledge is likely to be less important with more complex endgames where there are few known theorems determining the outcome and where, in general, a reliable database of perfect play is not available.

The most difficult stage of the entire process is almost certainly stage (ii) - finding a common property shared by most or all of a group of exceptions under consideration. Once this property is found it can, however, serve as the basis for a change to the algorithm. In searching for a common property, it will generally be necessary to look at both the set of positions arising after MOVEF in each position in the group of exceptions under consideration and the set of positions arising after MOVEP in each such position, on the assumption that either MOVEF is being consistently under-valued or MOVEP is being consistently over-valued.

In practice it is unlikely that any such fully consistent under - or over - valuation will be found and the aim should be to find a predicate function satisfied by a large proportion of the positions in one or other of the sets.

Although a suitable predicate, if it exists, may turn out to be any one of the possible values of <predicate> defined above, there may be a priori knowledge to suggest that some predicates are more likely to be useful than others. Thus for Pawn endings, "Pawn on Rook file" and "Pawn on second rank" are likely to be important, as is "Kings in opposition". More sophisticated knowledge may suggest other possible

interesting predicates such as "the Black King is closer to square C8 than the White".

A particular problem at this stage is avoiding finding a common property which is either too general (e.g. Pawn not on second rank) or too specific (e.g. Pawn on third rank and White King immediately in front of the Pawn, etc.). In the first case, it may happen that most or all of the positions satisfy the property purely by chance; in the second, the property may be satisfied by only a small number of positions out of the total group under consideration.

To help avoid these possibilities a large group of exceptions should be considered whenever possible and the probability of each predicate or combination of predicates occurring simply by chance should be taken into account.

For example, with the initial form of the algorithm for King and Pawn against King, 37% of the sample positions with a Rook Pawn were exceptions, compared with an overall level of exceptions of 11% for the whole sample (Section 10.2.1). This was a strong indication that it would be worthwhile to look at positions with a Rook Pawn in isolation from the others.

Of the 24 exceptions for which CLASSP=CLASSF=5 with the first revised form of the algorithm (Section 10.2.2), each of the successor positions arising after the "file move" had WP2=2, i.e. the Pawn was on the second rank, a result extremely unlikely to occur by chance. A further example, involving only a small number of positions, occurred when considering the exceptions remaining with the fifth revised version of the algorithm (Section 10.2.4). For the five exceptions for which CLASSP=CLASSF=15, the positions after each of the "file moves" are as follows, using the

notation of Table 25:

252746

656746

151736

555736

454726

The most striking feature of these simple co-ordinate representations of the positions is that the second, fourth and sixth co-ordinates are constant. This leads naturally to the description

WP2=6 AND WK2=5 AND BK2=7

A slightly less obvious relationship is that the first and third co-ordinates are equal for each position, i.e. WK1=BK1.

Finally, the difference between the first and fifth co-ordinates is always either 2 or -2

i.e. (WK1-WP1=2) OR (WK1-WP1=-2)

combining these gives trivially abs(WK1-WP1) = 2.

Taking the logical 'AND' of all five relationships gives the definition of equivalence class 19 in Section 10.2.4. A simple statistical analysis can be used to estimate the likelihood of a given predicate being true for a certain number of positions in a group purely by chance. Thus, if it is assumed for simplicity that the positions on the test file are a random sample of all possible positions, then since WK1 varies from 1 to 8 and WP1 varies from 1 to 4, the probability of the predicate abs(WK1-WP1)=2 being true by chance is 6/32 = 0.1875. The expectation from five positions would therefore be 0.9375 occurrences and the observed frequency of five occurrences would seem likely to be significant.

Although the above example was chosen for simplicity, a calculation of expectations is, of course, more likely to prove valuable in the case of a large group of positions, not all of which share the property under examination.

In general, when only small groups of exceptions remain, it may become extremely difficult to identify significant common properties (as it is for people) and it may then be unavoidable to introduce "special cases" to cater for only a few positions.

Provided that a property has been found for a particular group of exceptions, the nature of the group indicates the type of change (Stage (iii)) required to the algorithm. If the group contains positions with CLASSP = CLASSF at selection level 4, say, then a change to the third associated function is indicated.

In most cases, however, a new equivalence class or a change in the definition of an existing class will be required. The former should normally be chosen unless the full selection table shows that there is little or no support (in terms of correctly handled cases) for the class as it stands. A class of "positions to be avoided" should be inserted before CLASSP in testing order, with a value less than that of CLASSF. A class of "positions to be achieved" should be inserted before CLASSF in the testing order with a value greater than that of CLASSP.

Once a change to the algorithm has been chosen, making the change is fairly straightforward as has been discussed previously. Deciding on whether or not a change should be accepted (stage (iv)) is non-trivial for people although some heuristics can be applied, in particular the obvious one that if some exceptions have now vanished and no new ones have appeared (even if the details of MOVEP, CLASSP etc. have changed), the

change should be accepted. The main area of difficulty is undoubtedly the case where one group of exceptions has disappeared, but another has appeared and a further modification is needed before deciding whether or not to accept the first. It remains to be seen how difficult it is to discover the heuristics which people use in evaluating such situations.

11.4 Simplifying the algorithm

In implementing the above, the most likely problem to arise is that of failing to introduce equivalence classes at the right level of generality with the effect that a large number of over-specific classes are defined. It would therefore be helpful to include a separate "simplification stage" into the process of algorithm refinement, either at the end or at intermediate points in the process. The aim of this stage would primarily be to combine existing classes and to make the definition of existing classes more general. There are a number of rules which guarantee that classes can be safely combined (e.g. when they are consecutive in order of testing and none of their members are ever chosen as best). However, the process can better be performed on a heuristic basis, using "exception reporting" to validate any tentative changes made. This allows for the possibility of "inductive jumps", such as combining classes defined by

(i) WP2=4 AND WK2=WP2+1

(ii) WP2=6 AND WK2=WP2+1

and

(iii) WP2=3 AND WK2=WP2+1

into a single class, with definition simply

WK2=WP2+1.

Such a simplification stage, if effective, could mitigate the effect of fairly substantial weaknesses in implementing the remainder of the refinement process.

11.5 Discussion

In this section the possibility of implementing a fully-automated self-improving system of algorithm refinement has been considered.

Whilst the project as a whole remains an extremely substantial task, the components - looked at individually - do seem to be feasible. The most effective approach would no doubt be a process of gradual evolution from a partially-automated system of refinement such as that described in Section 10.

The method of algorithm simplification by synthesizing general rules from sets of more specific ones may have an important part to play in this process, together with recognising properties of groups of exceptions on an essentially probabilistic basis. To the extent to which implementing a self-improving system is feasible, the choice of underlying model would appear to be crucial. It is only because algorithm refinement using the equivalence class model appears to be fairly straightforward for people that it is worthwhile even to consider the possibility of automating the process, and the analysis in this section makes use of the characteristics of the model at virtually every stage.

One interesting possibility is that a self-refining system might build up an algorithm consisting of an efficient set of rules which were not at all natural from the viewpoint of a human chessplayer. Just as concepts such as "the opposition" were invented by chessplayers to summarise centuries of experience of playing endgames, and now

appear "natural", so there may be an entirely different set of concepts which could be derived from entirely different experience, by the process of rule synthesising described previously. However, the form of the final algorithm may be expected to depend critically on the initial algorithm provided, the basic primitives used for forming predicates and the rules governing the form of predicates.

Assuming that these all reflect the chessplayer's knowledge, as illustrated by the descriptions in standard textbooks, it seems unlikely that any radically different interpretations of the game will arise. Nevertheless, there is a possibility of extending existing knowledge of more complex endgames by discovering important new predicates. The nature of the equivalence class model as a structural rather than a procedural representation of knowledge enables any new predicates formed to be isolated and examined independently of the rest of the algorithm.

Although the possibility of developing a self-improving system has only been considered briefly here, it would seem that the choice of model is crucially important for such a project and that the equivalence class model appears to offer reasonable prospects for further investigation in this area.

12. Summary and discussion

This thesis began by describing a model which has been designed to facilitate the writing of algorithms which correspond reasonably closely to the human chessplayer's knowledge of the endgame, based on the descriptions given in textbooks. The algorithms have a simpler structure than those written using conventional models and can be described in a form meaningful to chessplayers themselves, with no specialist knowledge of computational techniques. An example of such a description is given in Appendix 1. As a means of illustrating some of the features of the model and of demonstrating its applicability in practice, algorithms for two relatively simple endgames, King and Rook against King and King and Pawn against King have been developed and subjected to substantial testing.

In principle, the same model could be used to construct algorithms for a wide range of different endgames, given some initial body of theoretical knowledge such as that which already exists in textbook form for most common endgames.

An example of the kind of algorithm which might be constructed for some of the situations which arise in a more complex endgame, King, Rook and Pawn against King and Rook, is given in Appendix 2. In cases where the basic form of the model proves inadequate, the extended form of the model described in Section 2.8 enables tree-searching to be included in a readily controllable manner.

Since virtually all endgames are understood only imperfectly, it is to be expected that 'initial' algorithms constructed from textbook descriptions will themselves contain inaccuracies and errors, to a greater or lesser extent. In addition, the translation of rules stated or implied in textbooks (including those given only implicitly, by

examples) into corresponding equivalence classes can at best be only an imprecise process. In order to construct powerful algorithms, it is necessary to combine the general model with some means of refining the initial algorithms produced.

For this reason, Sections 9-11 are devoted to a detailed discussion of the process of refining an algorithm for the endgame King and Pawn against King to play perfectly in each position on a given test file. This refinement process led to a much improved algorithm but one which retained the property of a close relationship to the chessplayer's endgame knowledge, within the framework of the equivalence class model. At each stage the form in which the exceptions were reported directly suggested the changes to be made. The final algorithm remained compact with no "special cases" of individual positions, and classes with relatively few members (such as class 19) were introduced only where there were insufficient exceptions remaining to enable more general specifications to be chosen.

When attempting to devise a systematic and generalisable method of program improvement it would seem likely that the choice of underlying model is critically important.

On the basis of the work described in Section 10 it appears that the equivalence class model lends itself naturally to iterative algorithm refinement, particularly since it is a structural rather than a procedural representation and the component elements of the algorithms are relatively simple and readily manipulable either by the programmer or an automatic system.

Thus, for example, to change the order of the goals held by the algorithm, it is only necessary to re-order the class values held in the first column of the value table. The possibility of extending the

techniques adopted in Section 10 into a fully automatic system of algorithm refinement was considered in Section 11, where the problem was broken down into its constituent parts. Once again it was found that the nature of the equivalence class model assisted at each stage of the refinement process and the principal outstanding problem was identified as that of synthesising a new class definition from an identified group of related exceptions. An approach to this problem based on the use of probability was suggested. Although the outstanding problems are no doubt substantial, various lines of investigation have been suggested and in general it would seem that the model chosen is a suitable one for future research in this area.

One interesting by-product of the work is that it calls into question the commonly held belief that endgames such as King and Pawn against King and King and Rook against King are completely understood theoretically and the knowledge contained (explicitly and implicitly) in textbooks is sufficient for correct (i.e. perfect) play in all positions. In fact, however, the examples given in Sections 6, 7 and 10 strongly suggest that this is not so. The process of algorithm refinement can therefore be considered as a possible means of extending existing knowledge within the same general framework.

This in turn raises the possibility that an algorithm of the kind under consideration - that is, capable of being understood and tested by chessplayers and given <u>advice</u> in the form of additional equivalence classes etc. - might be a superior alternative to the printed page as a medium for recording knowledge. The advantages of aiming for optimal play in a specified set of positions were stated in Section 10: firstly that the results achieved can be evaluated objectively and secondly that the requirements of a self-improving system can be studied in isolation from any possible errors or inconsistencies in the source data.

However, since it would seem that even for elementary endgames both chessplayers' and textbook knowledge do not suffice for perfect play it may be that such a criterion is an inappropriate aim for future work on knowledge representation. Even for elementary endgames and certainly for more complex ones it may be that a complete optimal strategy would occupy many pages of a textbook and would be an "unnatural" one for a chessplayer, involving a great deal of memorisation of specific positions. It would then be necessary to consider alternative definitions of an "optimal" algorithm, for example as one which is computationally efficient and finds a good move in every position and the best move in many cases.

12.1 Directions for further research

Some suggestions for further research have been given in previous sections.

These fall into two main categories: those related to the problem of constructing algorithms to play other endgames, and those related to more general problems in Artificial Intelligence.

With regard to the first of these, further experimentation is necessary with both the basic and the extended models to determine whether more complex endings and ultimately also middlegame positions can be satisfactorily handled. It remains to be seen whether or not the number of classes required increases disproportionately with each increase in the complexity of the endgame under investigation. However, given that the number of classes required continues to correspond fairly closely to the number of different situations recognized by the human chessplayer, there would seem to be no reason in principle why the same method should not be extended indefinitely. For endgames with several pieces it may, however, be necessary to introduce an initial pre-processing stage to reduce the number of moves (and thus successor positions) considered.

The second direction for research is of more general interest. The most obvious extension of the present work concerns the development of an automatic system for algorithm refinement, along the lines described in Section 11, in the case where a database of perfect knowledge is available. A further extension would then be to the more general case where only imperfect (and possibly conflicting or partly erroneous) knowledge is available. One possible way to extend the database of available knowledge would be to set up two algorithms, one for each side, and play them in competition to termination. A problem which falls somewhere between the two categories is to make a controlled study of the quality of human performance in simple endgames to establish realistic performance criteria for future research in this field, particularly in the case of endgames for which no database of perfect knowledge is available.

Finally, consideration should be given to applications of the equivalence class model to problem areas other than Chess to which state-space methods are normally applied.

Appendix 1. The move-finding algorithm for King and Rock against King: a non-technical description

This appendix contains a description of the move - finding algorithm (for the side with the Rook only) presented in a form in which it might be addressed to human chessplayers in a chess textbook, with the important difference that only the "recipe" for playing the end-game has been given, with no attempt at justification or explanation. The form of the algorithm to be described is the initial version, as given in Section 4.

The equivalence classes are represented by a series of diagrams, each with a brief accompanying description. Where there are associated functions for a particular class they are represented by an additional comment in parentheses alongside the diagram. Positions where the Rook is en prise, checkmate positions and stalemate positions are mentioned first but no diagrams are given to illustrate them. Diagrams for these classes could, of course, be easily included if required. A diagram is given for every other class (two diagrams for class 4), except class 11 which contains only "residual" positions. The number given to the top left of each diagram refers to the corresponding equivalence class and does not form part of the description itself.

The description of the move-finding algorithm now follows.

The endgame King and Rook against King

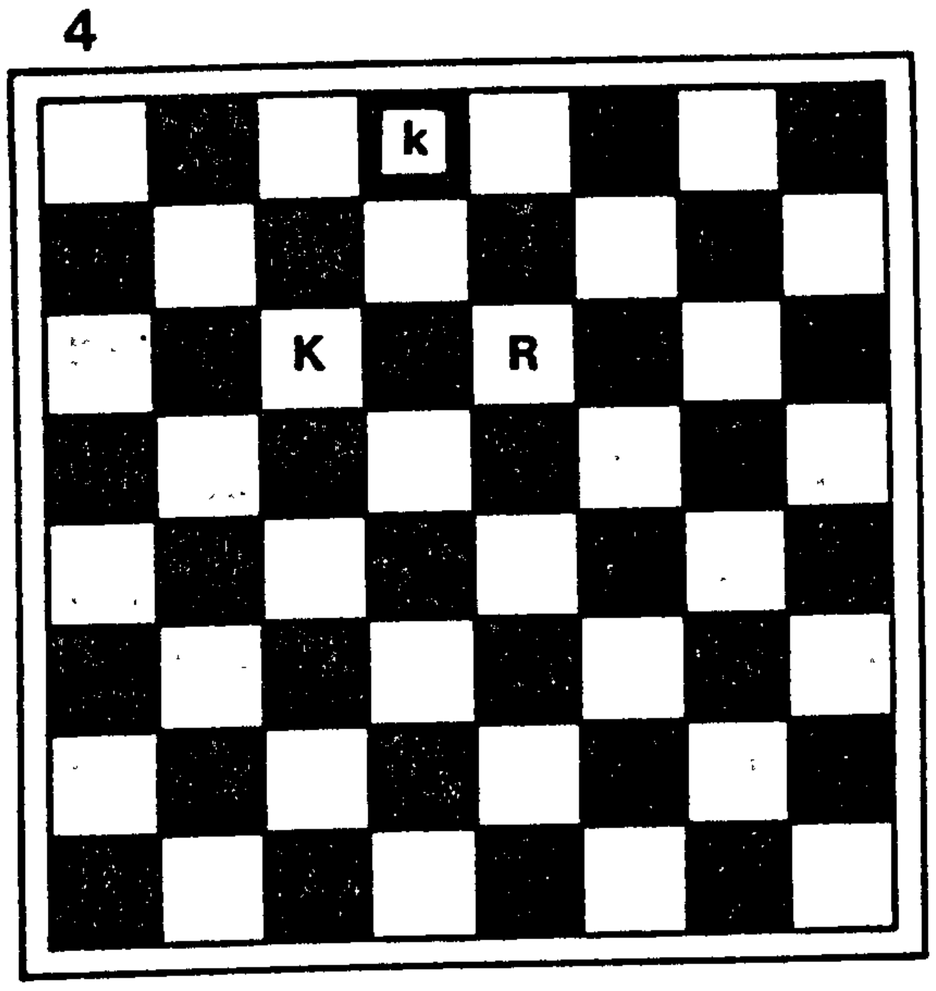
The winning strategy for White (the side with the Rook) can be summarised in the simple rule "always select the move which gives you the most favourable possible position". Since after any White move it is, of course, always Black's move in the resulting position, all that remains is to specify how to determine which is the best of a number of alternative possible positions with Black to move.

Clearly positions where Black is already checkmated are the best and positions where Black is stalemated or the Rook is en prise are the worst.

The following diagrams show the other types of position for which White should aim, arranged in descending order of preference.

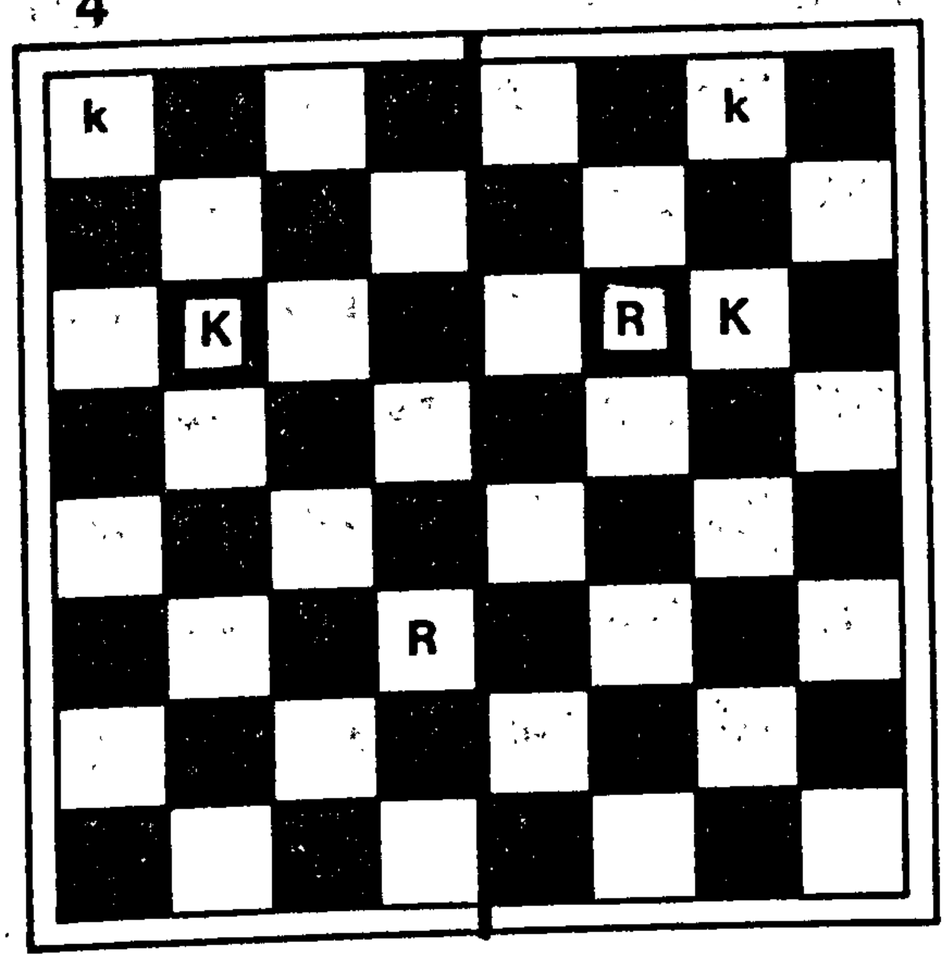
(If more than one diagram seems to apply to a position, the first one should be taken.)

4



Black cannot avoid mate in one.

4

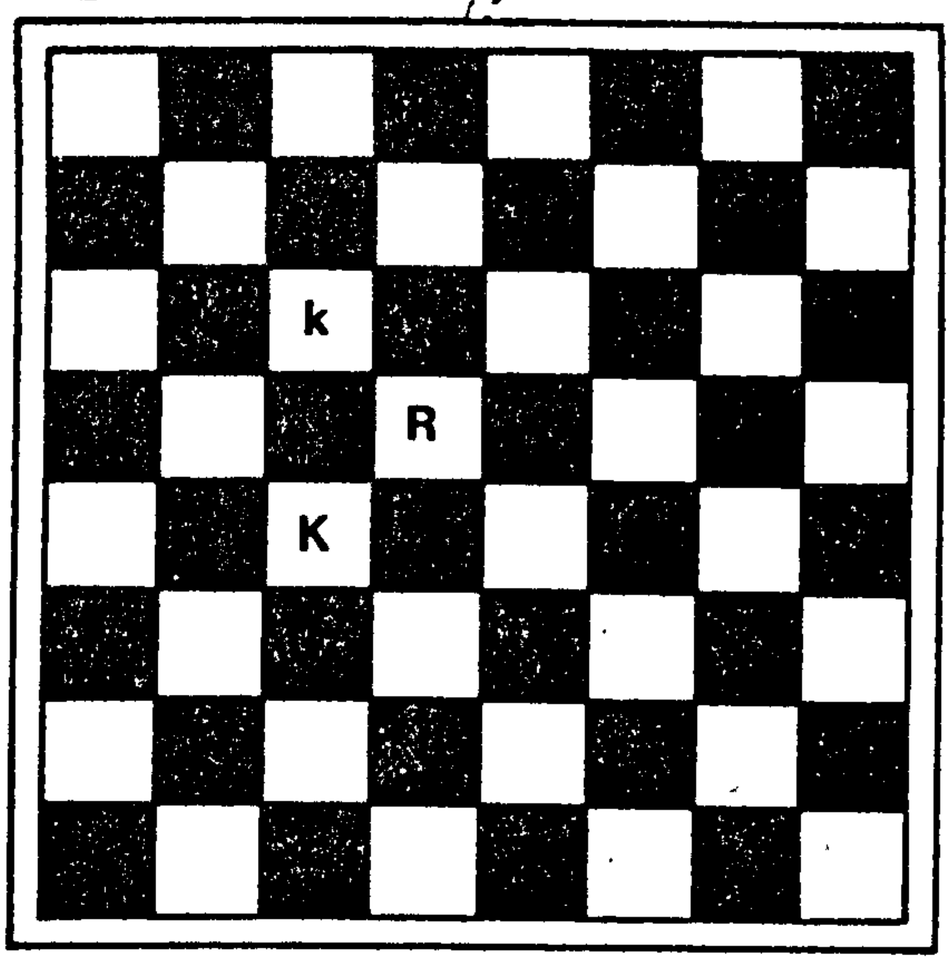


Black cannot avoid mate in one.

5

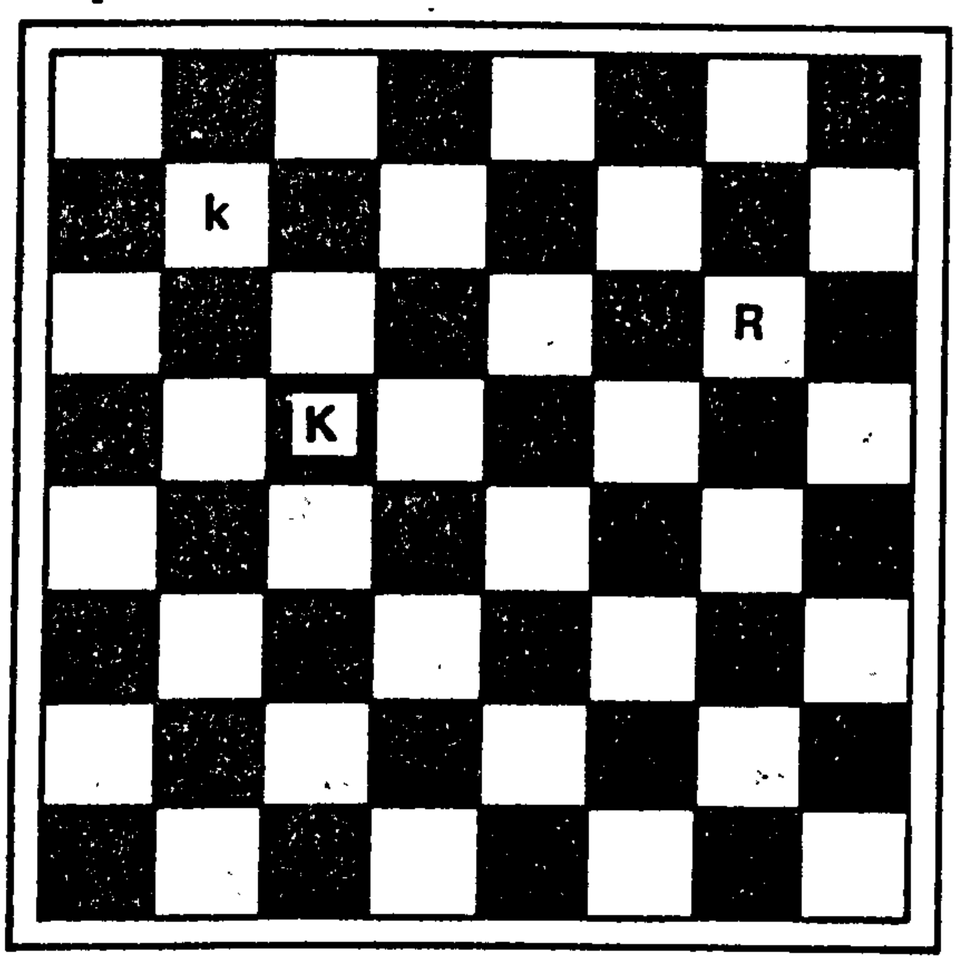
Kings in opposition with Black in check.

6



Kings in opposition with the Rook on the rank between them one file closer to the centre.

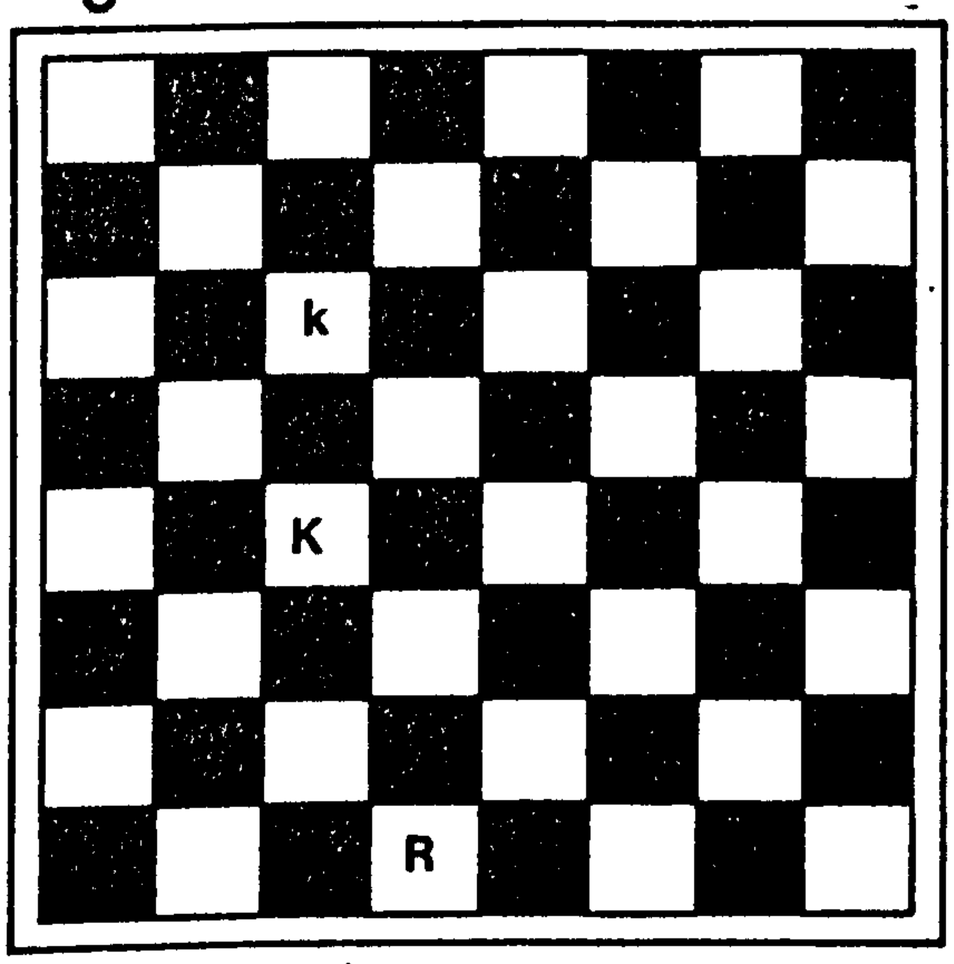
7



Kings two ranks apart, with White one file closer to the centre and the Rook on the rank between them. The Rook must not be on either of the files adjacent to White's King.

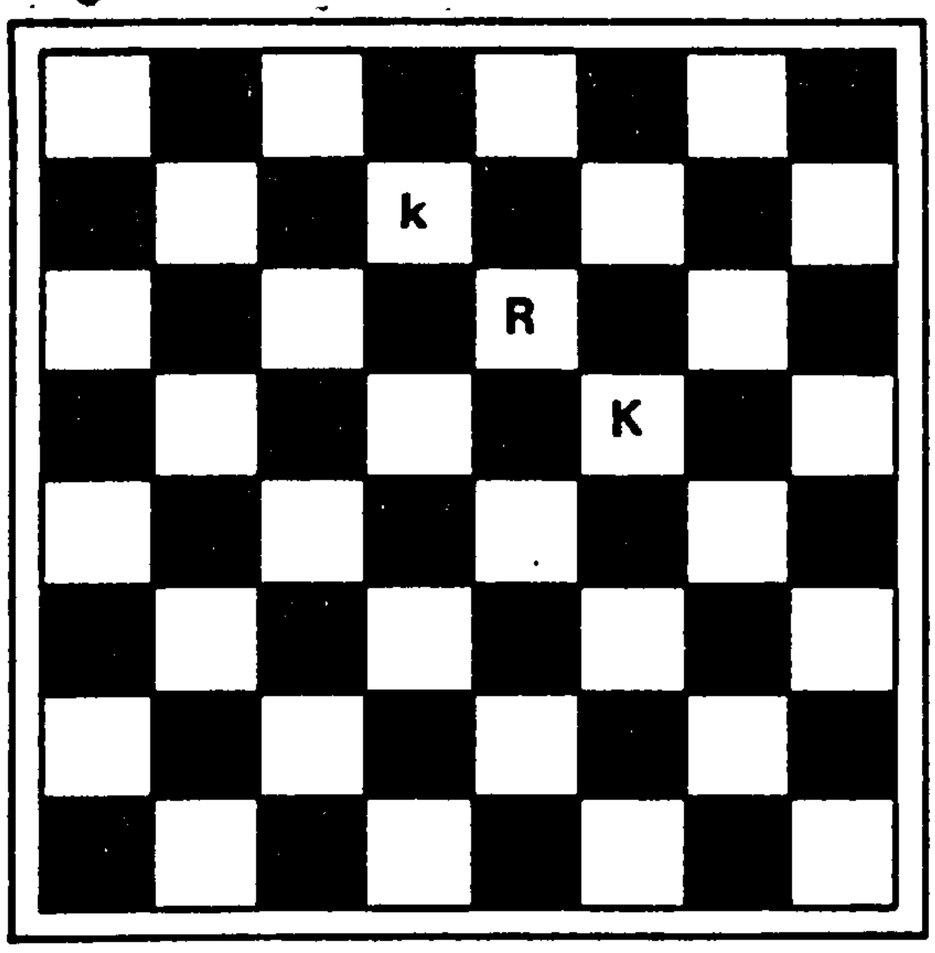
(Try to restrict Black's King to as small a quadrant as possible.)

8



Kings in opposition with Rook one file closer to the centre.

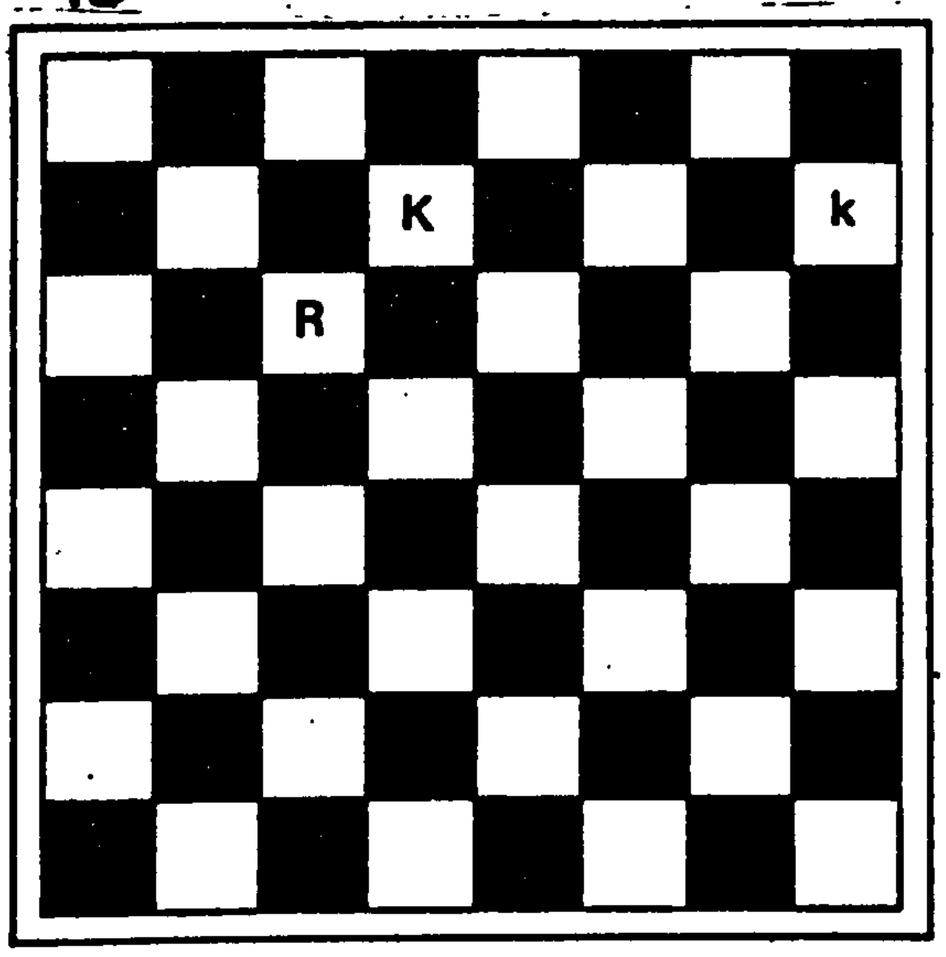
9



Kings two ranks apart, with the Rook on the rank between them. The Black King must not be closer to the Rook than the White King is.

(Try to restrict Black's King to as small a quadrant as possible. Subordinate to that, the two Kings should be as close together as possible.)

10



Black King inside a quadrant set up by the action of the Rook. The White King can be anywhere except on the boundary of the quadrant when this would allow Black to escape.

(Try to restrict the Black King to the smallest possible number of ranks. Subordinate to that, the two Kings should be as close together as possible.)

Appendix 2. King, Rook and Pawn against King and Rook:

Steps towards the development of an algorithm

This endgame is considerably more complicated than the other two discussed in this thesis. Whereas Fine (1941) devotes less than a page to King and Rook against King, the description of King, Rook and Pawn against King and Rook occupies thirty pages. The aim of this appendix is to indicate the kind of initial form of algorithm which can be derived from the examples and analysis given by Fine.

For this purpose two short extracts will be considered, relating to two critical positions for this ending known as Philidor's position and the Lucena position. (Both extracts make use of the "descriptive" system of notation.) In each case, a fairly simple set of equivalence classes is sufficient to produce the play given in the text. These are to be considered as part of two much larger sets of equivalence classes for this ending, one (of positions with Black to move) to be used to play White's strategy and another (of positions with White to move) to be used for Black's strategy. White is the side with the Pawn throughout. It should be noted that in both cases, equivalence classes such as "Black's Rook is en prise" and "Black Rook on the same square as the Pawn (i.e. has just taken the Pawn)" are assumed to be included, with appropriate class values, early in the testing order. These will not be explicitly mentioned further.

The notation WK1,WK2,BK1,BK2,WR1,WR2,BR1,BR2,WP1,WP2 is used to represent the file and rank co-ordinates of the White King, the Black King, the White Rook, the Black Rook and the White Pawn, respectively.

1. Philidor's position

ROOK AND PAWN VS. ROOK

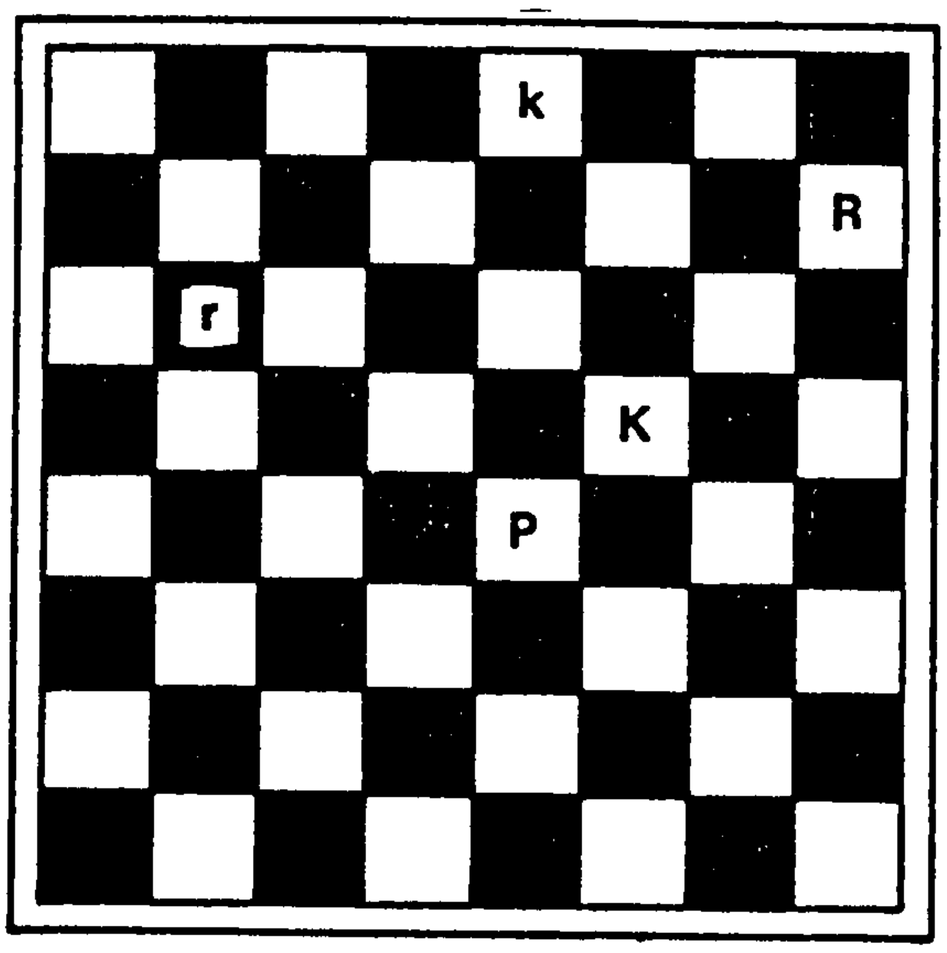
General rule: If the Black King can reach the queening square, the game is drawn; if not the game is lost. While this holds in most cases, it must be looked upon as a convenient rule of thumb and not adhered to too rigidly. In particular, the RP forms an exception to the second part.

Since it is difficult to grasp the essence of this ending without knowing a number of examples, and since the ending is so basic in all R and P play, we shall give a fairly exhaustive analysis. To begin with, we shall give the ideal drawing position, according to our general rule.

A. THE BLACK KING IS ON THE QUEENING SQUARE

If the Pawn is not far advanced, the King cannot be driven away. This has been known since the time of Philidor. No. 303 is the standard position. Black keeps his R on the third Rank until the P reaches the sixth, and then goes to the eighth. When the square K6 is no longer available for the White King he will not be able to occupy Q6 or KB6 and will be unable to drive the Black King out. The moves might be 1 P-K5; R-QR3; 2 P-K6, R-R8 (2 .... R-Kt3?? would be a bad blunder - 3 K-B6, K-Q1; 4 R-R8ch, K-B2; 5 K-B7 wins); 3 K-B6, R-B8ch; 4 K-K5, R-K8ch; 5 K-Q6, R-Q8ch, etc.

No. 303



Draw

The following classes are sufficient to produce correct play for Black in the example given above. Since moves for Black are to be found, these are all classes of position <u>with White to move:</u>

The predicate "Philidor" is used to represent (BK1=WP1) AND (EK2=8) AND (WR2=7).

Class	Definition (White to move)	Value	First associated Function
a	Philidor AND (BR2=6) AND (WP2<6)	4	abs (WK1-BR1)
b	Philidor AND (BR1=WP1) AND(BR2<WP2) AND dist (WK1,WK2,WP1,WP2)>2	3	
c	Philidor AND (P2=6) AND (BR1=WK1) AND (BR2<WK2)	2	
d	Philidor	1	8-BR2

With these classes and with White choosing the moves given by Fine, the main line from the diagrammed position is

1. P-K5 R-QR3 (a)
2. P-K6 R-R8 (d)
3. K-E6 R-B8ch (c)
4. K-K5 R-K8ch (c)
5. K-Q6 R-Q8ch (c)

etc.

With the figure in parentheses after each Black move giving the class of the position which results after that move (White to move). Class b is included to allow for sequences such as

6. K-B7 R-B8ch (c)
7. K-N6 R-K8 (b)

winning the Pawn.

Some other possible variations are

(i) 1. K-K5 R-QR3 (a)

or

(ii) 1. P-K5 R-QR3 (a)

2. P-K6 R-R8 (d)

3. K-B6 R-B8ch (c)

4. K-N7

here Black should play 4. ... K-K2. To ensure this, a further class (e) could be defined, with value 5 and definition BK1=WP1 AND BK2>WP2 with associated functions 8-BK2 and abs (WK1-BR1)

Including this class is equivalent to Black following the advice "occupy the file in front of the Pawn, with your King as close to the Pawn and your Rook as far from the White King as possible".

A further possible variation would then be:

1. R-R1 K-K2 (e)

2. R-K1 R-QR3 (e)

and White is clearly making no progress.

2. The Lucena Position

THE LUCENA POSITION

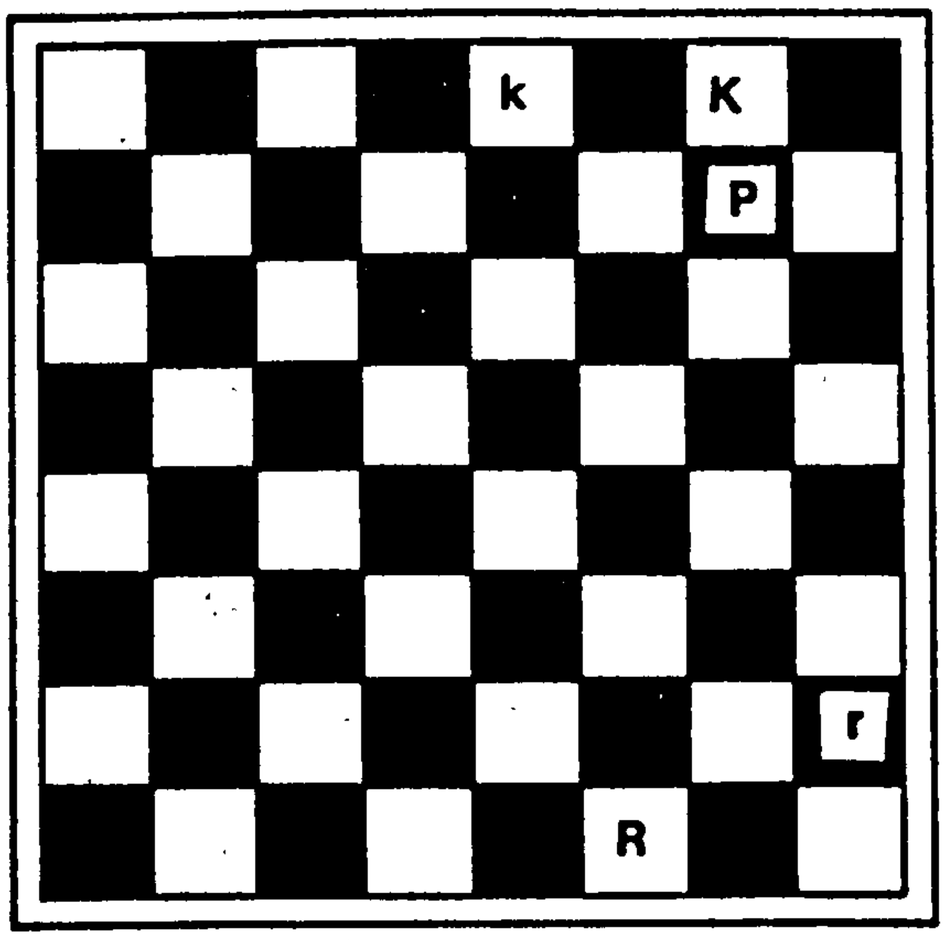
This is the key to all these endings. It was first discovered by Lucena, a Spanish author who wrote towards the end of the fifteenth century. The solution to No. 307 is 1 .... R-R6; 2 R-B4! ("building a bridge" is an apt description of the winning method), R-R8; 3 R-K4ch, K-Q2; 4 K-B7, R-B8ch; 5K-Kt6, R-Kt8ch; 6 K-B6, R-B8ch (if 6,... R-Kt7; 7 R-K5 followed by R-KKt5, while if 6.... K-Q3; 7 R-Q4 ch, K-B3; 8 R-Q8, R-B7ch; 9 K-K5, R-K7ch; 10 K-B4, etc.; 7 K-Kt5, R-Kt8ch; R-Kt4 and wins.

If the Black R leaves the R file, say 1.... R-K7, then 2 R-KR1 allows the exit of the White K to KR8 or KR7, freeing the P. If the Black K goes to K2 and B3, the White K gets to B8, e.g. 1 .... K-K2; 2 R-K1ch, K-B3 (2 .... K-Q3; 3 R-K4 as above); 3 K-B8.

Clearly, these lines are not applicable to a RP, since there the exit of the White K is blocked on the one side he can go to by both K and R.

The Lucena Position

No.307



White wins; win in all analogous positions except with RP.

Here it is White's strategy which is of interest. Clearly the position after his second move in the above sequence (with the Rook on the fourth rank) is particularly important. The following equivalence classes include one defining this position explicitly.

The predicate "Lucena" is used to represent (WK2=BK2=8) AND (WP2=7) AND (WK1=WP1) AND (WK1=BK1+2) AND (BR1=WK1+1) AND (WR2=4).

Class	Definition (Black to move)	Value	Associated functions	
			First	Second
a	(BR1=WR1=WK1=WP1) AND (BR2<WR2< WK2<WP2) AND (WP2=7) AND dist (BK1,BK2,WP1,WP2)>2	4		
b	(WR1=BK1+1) AND (WP1≥BK1+3) AND WP2 =7 AND(WK1=WP1 OR WK1=WP1-1)	3	8-K2	8-abs (WK1-WP1)
c	Lucena AND WR1=BK1	2		
d	Lucena AND WR1=BK1+1	1		

(Note that it has been assumed throughout that BK1<WK1. This can be thought of as a standard orientation for these classes.)

Using these classes, the main variation given by Fine will be correctly treated. The figures in parentheses below show the class of which the position after each White move is a member.

1.	...		R-R6
2.	R-B4	(d)	R-R8
3.	R-K4ch	(c)	K-Q2
4.	K-B7	(b)	R-B8ch
5.	K-N6	(b)	R-N8ch

6. K-B6 (b) R-B8ch

7. K-N5 (b) R-N8ch

8. R-N4 (a)

and wins.

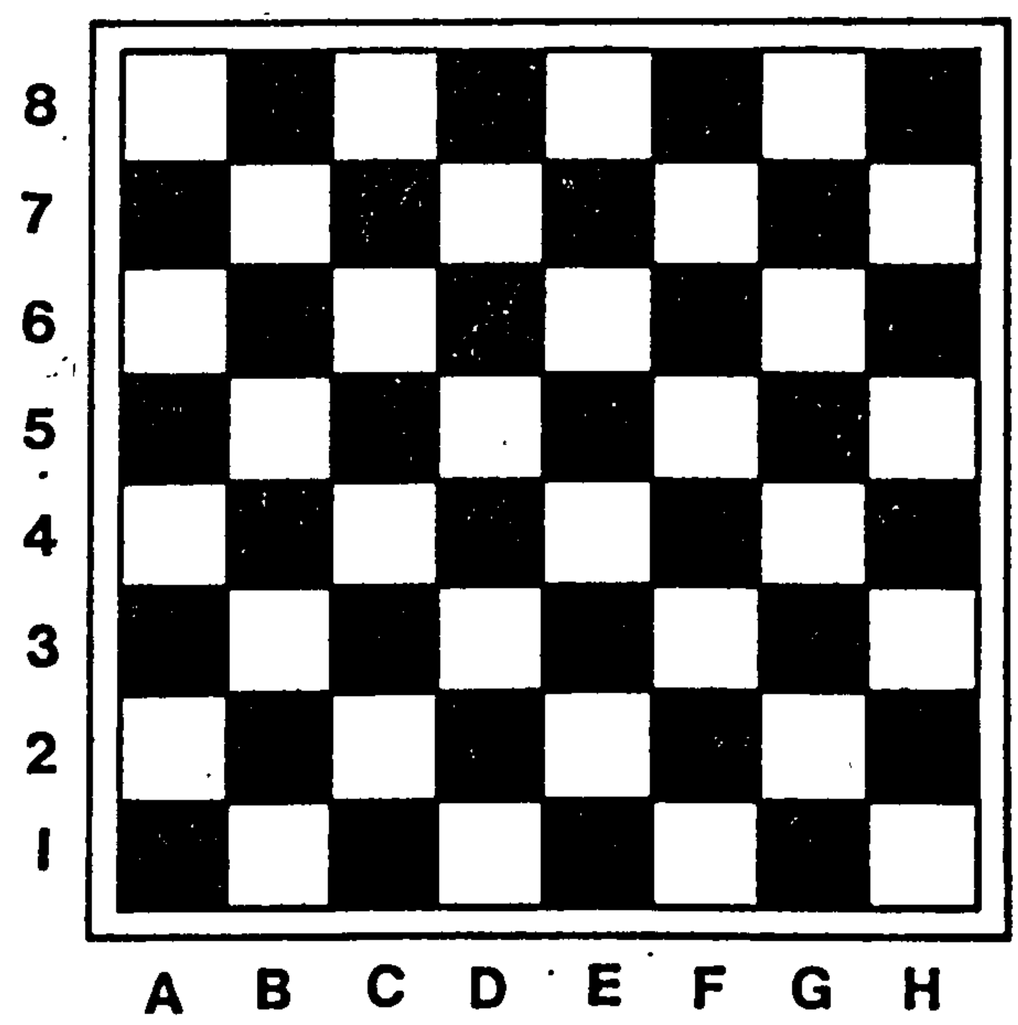
On examining the two extracts from Fine, it is clear that from the viewpoint of the human chessplayer, there seem to be several different important positions and details given and that some of these appear to be highly specific to particular cases only.

The equivalence classes given above are a reflection of this situation. Although the classes as defined appear to be an appropriate choice for the examples given, they may ignore some important exceptional cases, or alternatively be themselves only specific instances of more general classes of positions. For example, in the definition of the predicate "Lucena", it may be that the Rook could be on the fifth or sixth ranks instead of the fourth.

Nevertheless, provided that an initial version of an algorithm can be set up, the examples given in textbooks and occurring in master play can be used to refine it, either "by hand" or perhaps ultimately by a fully automatic method, as discussed in Section 11.

Appendix 3. Notation

(1) Squares are represented using algebraic notation. The files are designated by a letter A to H, from left to right, and the ranks are numbered 1 to 8, from bottom to top of the board. Each square is then uniquely represented by a letter followed by a number, e.g. A8. (It is assumed that the board is arranged with White's first rank at the bottom.)



(2) Moves are denoted by a letter, representing the piece moved, followed by a hyphen, followed by the destination square, e.g. R-E6. When it is necessary to distinguish between a White move and a Black move, the latter is preceded by three dots, e.g. ...K-A2.

(3) WK1 and WK2 are used to denote the number of the rank and file of the square occupied by the White King. Thus, if the King is on square A6, WK1 is 1 and WK2 is 6.
BK1 and BK2, WR1 and WR2 and WP1 and WP2 are used similarly for the Black King, the White Rook and the White Pawn.

(4) In diagrams, "K", "k", "R" and "P" are used to represent the White King, the Black King, the White Rook and the White Pawn, respectively. The player to move is indicated by the letter W or B to the top left of the diagram.

(5) Function names are printed in lower case letters and underlined, except when they consist of only one letter.

(i) abs - absolute value function; abs (-3) = abs (3) = 3.

(ii) max - "maximum" function; max (6, -4) = 6

(iii) min - "minimum" function; min (6, -4) = -4

(iv) dist - the "block distance" between two squares e.g. dist (WK1,WK2,BK1,BK2) is given by max (abs (WK1-BK1), abs (WK2-BK2)). When both squares are occupied by pieces, the arguments of the function are generally replaced by the names of the two pieces, for convenience. Thus, the above would be written as dist (White King, Black King).

(6) AND, OR and NOT are the customary logical connectives ('OR' is inclusive).

(7) The notation introduced in Section 2 is summarized below.

W - the set of legal positionswith White to move for a given endgame

B - the set of legal positions with Black to move for a given endgame

p - an initial position with White to move

Q(p) - the (ordered) set of immediate successors of p (sometimes abbreviated to Q)

$q_i(p)$ - the i'th immediate successor of p (sometimes abbreviated to q_i or q)

C_i - the i'th equivalence class of positions

C(b) - the equivalence class of which position b is a member, $b \in B$.

V_i - the value of class C_i

V(b) - the value of class C(b)

f_{ji} - j'th associated function for class C_i (sometimes abbreviated to f_j)

N - the number of equivalence classes

m - the number of associated functions for each class

K - an integer value larger than the maximum value of any associated function for any class

(8) MOVEP and MOVEF are the 'program move' and 'file move', respectively, in a given position, i.e. the move selected by the move-finding algorithm for a particular endgame and the move stored on a database of best (or believed best) moves.

(9) CLASSP and CLASSF are the equivalence classes of which the positions which arise after playing MOVEP and MOVEF in a given initial position are members.

References

Awerbach, J. Lehrbuch der Endspiele, Vol. 1. Bauernendspiele, Berlin , 1958.

Bramer, M.A. King and Pawn against King : Some quantitative Data. Open University, Faculty of Mathematics, Technical Report, 1977.

Capablanca, J.R. A Primer of Chess. Harcourt Brace, New York, 1935.

Clarke, M.R.B. The Computational Complexity of King and Pawn Versus King, 1975, Mimeo. Forthcoming in Clarke, M.R.B. (ed.), Advances in Computer Chess 1. Edinburgh University Press.

Elcock, E.W. Descriptions. In Michie, D. (ed.), Machine Intelligence 3, Edinburgh University Press, 1968, 173-179.

Elcock, E.W. and Murray, A.M. Experiments with a Learning Component in a Go-Moku Playing Program. In Collins, N.L. and Michie, D. (eds.), Machine Intelligence 1, Oliver and Boyd, Edinburgh and London, 1967, 87-10

Fine, R. Basic Chess Endings. David McKay, New York, 1941.

Fine, R. Chess the Easy Way. David McKay, New York, 1944.

Golombek, H. The Game of Chess. Penguin, Harmondsworth, 1954.

Hayes, J.E. and Levy, D.N.L. The World Computer Chess Championship. Edinburgh University Press, 1976.

Howe, J.A.M. et al. Artificial Intelligence and the Representation of Knowledge. University of Edinburgh, Department of Artificial Intelligence, Research Report No. 5, 1975.

Huberman, B.J. A Program to Play Chess Endgames. Ph.D. dissertation, Stanford University, 1968.

Lasker, Ed. The Adventure of Chess. Dover, New York, 1959.

Michie, D. AL1: A Package for Generating Strategies from Tables. University of Edinburgh, Machine Intelligence Research Unit, Memorandum MIP-R-114, 1976b.

Michie, D. King and Rook against King. 1. Historical Background and a Problem on the Infinite Board. 1976a, Mimeo. Forthcoming in Clarke, M.R.B. (ed.), Advances in Computer Chess 1. Edinburgh University Press.

Murray, A.M. and Elcock, E.W. Automatic Description and Recognition of Board Patterns in Go-Moku. In Dale, E. and Michie, D. (eds.), Machine Intelligence 2, Oliver and Boyd, Edinburgh and London, 1968, 75-88.

Newell, A., Shaw, J.C. and Simon, H.A. Chess-playing Programs and the Problem of Complexity. IBM Journal of Research and Development, October 1958, 2, 320-335. Reprinted in Feigenbaum, E.A. and Feldman, J. (eds.), Computers and Thought, McGraw-Hill, 1963, 39-70.

Nimzowitsch, A. My System . Bell, London, 1929.

Popplestone, R.J. An Experiment in Automatic Induction. In Meltzer, B. and Michie, D. (eds.), Machine Intelligence 5, Edinburgh University Press, 1969, 203-215.

Samuel, A.L. Some Studies in Machine Learning using the Game of Checkers. IBM Journal of Research and Development, July 1959, 3, 211-229. Reprinted in Feigenbaum, E.A. and Feldman, J. (eds.), Computers and Thought, McGraw-Hill, 1963, 71-105.

Samuel, A.L. Some Studies in Machine Learning using the Game of Checkers II. Recent Progress. IBM Journal of Research and Development, 1967, 11, 601-617.

Shannon, C.E. Programming a Computer for Playing Chess, Philosophical Magazine, 1950, 41(314), 256-275.

Smith, M.H. A Learning Program which Plays Partnership Dominoes. Communications of the A.C.M., 1973, 16(8), 462-467.

Tan, S.T. Kings, Pawn and Bishop. University of Edinburgh, School of Artificial Intelligence, Memorandum MIP-R-108, 1974.

Tan, S.T. Representation of Knowledge for Very Simple Pawn Endings in Chess. University of Edinburgh, School of Artificial Intelligence, Memorandum MIP-R-98, 1972.

Vigneron, H. Les Automates, La Nature, 1914, 56-61.

Waterman, D.A. An Introduction to Production Systems.
AISB European Newsletter, January 1977, 25, 7-10.

Winston, P.H. Learning Structural Descriptions from Examples.
Massachusetts Institute of Technology, Artificial Intelligence
Laboratory, A.I. Technical Report 231, 1970.

Winston, P.H. The MIT Robot. In Meltzer, B. and Michie, D. (eds.),
Machine Intelligence 7, Edinburgh University Press, 1972, 431-463.

Young, R.M. Production Systems as Models of Cognitive Development.
University of Edinburgh, School of Artificial Intelligence -
Bionics Research Laboratory, Bionics Research Report No. 22, 1974.

Zuidema, C. Chess, How to Program the Exceptions?
Stichting Mathematisch Centrum, Ze Boerhaavestraat 49
Amsterdam, 1974.

King and Pawn against King: Some quantitative data

M.A. Bramer

January, 1977

CONTENTS

1. Introduction

This report contains a number of tables giving quantitative data for the chess endgame King and Pawn against King, gathered mainly in the course of the work described in Bramer (1977).

Although this data is of some interest in itself, it is hoped that it will also prove useful to those involved in research based on this endgame and particularly in conjunction with work on the measurement of a program's "knowledge content".

All the data has been obtained either by means of simple counting programs or by analysis of a database set up and made available by Michael Clarke of Queen Mary College, London. The database is described in detail in Clarke (1975).

The data given here is by no means complete and the author will be grateful to receive any additional information or independent corroboration of any of the values given.

Acknowledgements

I should like to thank Michael Clarke for the use of his King and Pawn against King database and Don Beal, of Queen Mary College, for making an independent verification of some of the principal figures.

2. Standard board orientation and definition of terminal positions

For convenience, the conventions followed in Clarke (1975) are also adopted here.

The side with the Pawn will be assumed to be White throughout and for reasons of symmetry, the Pawn will be restricted to files A to D only (thus each position corresponds to two in the full state space).

It is assumed that White wins by advancing his Pawn to the eighth rank, provided that it cannot immediately be captured. (This includes positions where promotion to a Rook is necessary to avoid stalemate). For this reason, positions with the Pawn on the eighth rank and White to move are excluded from consideration, as are positions where the Pawn is off the board (captured by Black). Positions where the Pawn is on the eighth rank with Black to move are included,however.

The two positions where White to play is stalemated are excluded but the nine positions where Black to move is stalemated are included.

The following types of position are considered terminal, with Black to move in each case:

(a) Win for White

the Pawn is on the eighth rank and not en prise.

(b) Draw

(i) Black is stalemated.

(ii) the Pawn is en prise.

(iii) Black can stalemate White in one move (there are seven such positions).

In passing, it will be noted that all positions with the Pawn on the eighth rank (Black to move) are terminal.

2.1 Breakdown of all legal positions

Number of legal positions	White to move 81662*	Black to move 97992**
Number of wins for White	62480(76.5%)	61787(63.1%)
Number of draws	19182*(23.5%)	36205**(36.9%)

* = excluding 2 stalemates

** = including 9 stalemates

2.2 Breakdown of legal positions by Pawn's square

Breakdown of legal positions by Pawn's square
WHITE TO MOVE

7	3436	3384	3386	3386	13592
6	3441	3389	3392	3392	13614
5	3441	3389	3392	3392	13614
4	3441	3389	3392	3392	13614
3	3441	3389	3392	3392	13614
2	3441	3389	3392	3392	13614
	A	B	C	D	
	20641	20329	20346	20346	81662

Breakdown of legal positions by Pawn's square
BLACK TO MOVE

8	3492	3496	3496	3496	13980
7	3496	3502	3502	3502	14002
6	3496	3502	3502	3502	14002
5	3496	3502	3502	3502	14002
4	3496	3502	3502	3502	14002
3	3496	3502	3502	3502	14002
2	3496	3502	3502	3502	14002
	A	B	C	D	
	24468	24508	24508	24508	97992

2.3 Breakdown of wins by Pawn's square

WHITE TO MOVE

7	3321	3229	3240	3240	13030
6	3054	2962	2894	2922	11832
5	2707	2639	2574	2558	10478
4	2308	2369	2352	2395	9424
3	1887	2186	2219	2337	8629
2	1840	2316	2395	2536	9087
	A	B	C	D	
	15117	15701	15674	15988	62480

BLACK TO MOVE

8	3321	3218	3223	3223	12985
7	3052	2924	2794	2804	11574
6	2690	2573	2446	2355	10064
5	2260	2169	2029	1928	8386
4	1790	1835	1711	1663	6999
3	1307	1569	1507	1456	5839
2	1214	1575	1570	1581	5940
	A	B	C	D	
	15634	15863	15280	15010	61787

2.4 Breakdown of draws by Pawn's square

WHITE TO MOVE

	A	B	C	D	
7	115	155	146	146	562
6	387	427	498	470	1782
5	734	750	818	834	3136
4	1133	1020	1040	997	4190
3	1554	1203	1173	1055	4985
2	1601	1073	997	856	4527
	5524	4628	4672	4358	19182

BLACK TO MOVE

	A	B	C	D	
8	171	278	273	273	995
7	444	578	708	698	2428
6	806	929	1056	1147	3938
5	1236	1333	1473	1574	5616
4	1706	1667	1791	1839	7003
3	2189	1933	1995	2046	8163
2	2282	1927	1932	1921	8062
	8834	8645	9228	9498	36205

2.5 Terminal positions (Black to move only)

Win for White	12985	
Pawn en prise	10093	
Stalemate	9	
Black can stalemate White	7	
Total terminal positions	23094	(23.6%)
Total non-terminal positions	74898	(76.4%)
Total positions	97992	(100%)

2.6 Terminal v. non-terminal wins for White (Black to move only)

Terminal wins	12985	(21.0%)
Non-terminal wins	48802	(79.0%)
Total wins	61787	(100%)

2.7 Terminal v. non-terminal draws (Black to move only)

Terminal draws	10109	(27.9%)
Non-terminal draws	26096	(72.1%)
Total draws	36205	(100%)

2.8 Breakdown of legal positions with Pawn on eighth rank (Black to move only)

Pawn en prise (terminal draws)	995	(7.1%)
Others (terminal wins)	12985	(92.9%)
Total	13980	(100%)

2.9 Breakdown of Pawn en prise positions by Pawn's square (Black to move only)

	A	B	C	D	
8	171	278	273	273	995
7	278	425	417	417	1537
6	273	417	408	408	1506
5	273	417	408	408	1506
4	273	417	408	408	1506
3	273	417	408	408	1506
2	278	425	417	417	1537
	1819	2796	2739	2739	10093

2.10 Stalemate positions (Black to move only)

WK	BK	WP
A6	A8	A7
B6	B8	B7
C6	C8	C7
D6	D8	D7
B6	A8	A7
A6	A8	C7
B6	A8	C7
C7	A8	B6
C8	A8	B6

2.11 Positions from which Black to move can stalemate White

WK on A8, WP on A7 and BK on B6, C6, C7, C8, D6, D7 or D8.

3. Classification of won positions with White to move

Positions in which White to play has a forced win against any defence by Black can be classified into the following four disjoint and exhaustive types.

Type 0

Positions where the Black King is outside the queening square of the Pawn and the White King is not in front of the Pawn. In such cases the Pawn can 'run'. If it is on rank P2, then White wins in 8-P2 moves ($2 \leq P2 < 8$), or 5 moves (beginning with a double move) if P2 = 2.

Note that a Pawn on the second rank is considered to be on the third rank for purposes of constructing the queening square.

Type 1

Positions not included in type 0, where the Pawn can still 'run' (because the Black King's advance to cut off the Pawn is blocked or prevented in some way). As for type 0, White wins in 8-P2 moves ($2 < P2 < 8$) or 5 moves (P2 = 2).

Type 2

Positions not included in types 0 or 1, where White has a single best winning move (i.e. one which wins at a shorter depth than any other against best play by Black).

Type 3

Positions where White has two or more equal 'best' winning moves. (Note that positions in types 0 or 1 cannot fall into this category).

3.1 Breakdown of positions with White to move by Pawn's file

	A	B	C	D	TOTAL
Type 0 wins	13519	12214	10978	10512	47223
Type 1 wins	318	456	498	450	1722
Type 2 wins	269	1111	1616	1837	4833
Type 3 wins	1011	1920	2582	3189	8702
Total wins	15117	15701	15674	15988	62480
Draws	5524	4628	4672	4358	19182
Total legal positions	20641	20329	20346	20346	81662

3.2 Distribution of type 0 wins with White to move by Pawn's square

7	3263	3155	3160	3160	12738
6	2937	2779	2620	2628	10964
5	2519	2313	2106	1899	8837
4	2015	1763	1510	1243	6531
3	1431	1135	821	821	4208
2	1354	1069	761	761	3945
	A	B	C	D	
	13519	12214	10978	10512	47223

No of moves to win
1
2
3
4
5
5

3.3 Distribution of type 1 wins with White to move by Pawn's square

7	0	4	4	4	12
6	5	20	28	30	83
5	23	46	60	86	215
4	61	90	100	128	379
3	115	148	154	102	519
2	114	148	152	100	514
	A	B	C	D	
	318	456	498	450	1722

No of moves to win
1
2
3
4
5
5

3.4 Distribution of type 2 wins with White to move by Pawn's square

7	3	8	12	12	35
6	4	34	61	68	167
5	14	77	113	146	350
4	52	213	304	391	960
3	88	343	518	566	1515
2	108	436	608	654	1806
	A	B	C	D	
	269	1111	1616	1837	4833

3.5 Distribution of type 3 wins with White to move by Pawn's square

7	55	62	64	64	245
6	108	129	185	196	618
5	151	203	295	427	1076
4	180	303	438	633	1554
3	253	560	726	848	2387
2	264	663	874	1021	2822
	A	B	C	D	
	1011	1920	2582	3189	8702

3.6 Number of positions with White to move with N equal best winning moves

N	number of positions	
1	53778	(types 0, 1 and 2)
2	4115	type 3
3	2514	
4	870	
5	507	
6	384	
7	312	
8	0	
9	0	
10	0	
TOTAL	62480	

4. Classification of won positions with Black to move

Non-terminal positions in which Black to move cannot avoid defeat against best play by White can be classified into the following three disjoint and exhaustive types.

Type 0

Positions where the Black King is outside the queening square of the Pawn by more than one rank or file and the White King is not in front of the Pawn. In such cases, whatever move Black makes, the Pawn can 'run'.

If it is on rank P2, then White wins in 8-P2 moves ($2 \leq P2 \leq 8$) or 5 moves (beginning with a double move) if P2 = 2.

Note that a Pawn on the second rank is considered to be on the third rank for purposes of constructing the queening square.

Type 1

Positions not included in type 0, where after any move by Black the Pawn can still 'run' (because the Black King's advance to cut off the Pawn is blocked or prevented in some way).

As for type 0, White wins in 8-P2 moves ($2 \leq P2 \leq 8$) or 5 moves (P2 = 2).

Type 2

Positions not included in types 0 or 1.

4.1 Breakdown of positions with Black to move by Pawn's file

	A	B	C	D	TOTAL
Type 0 wins	10519	8817	7868	7393	34597
Type 1 wins	555	740	585	482	2362
Type 2 wins	4560	6306	6827	7135	24828
Total wins	15634	15863	15280	15010	61787
Draws	8834	8645	9228	9498	36205
Total legal positions	24468	24508	24508	24508	97992

4.2 Distribution of type 0 wins with Black to move by Pawn's square

	A	B	C	D		Number of moves White needs to win
8	0	0	0	0	0	0
7	2990	2828	2666	2674	11158	1
6	2565	2354	2143	1932	8994	2
5	2052	1794	1536	1264	6646	3
4	1457	1154	834	834	4279	4
3	786	420	420	420	2046	5
2	669	267	269	269	1474	5
	10519	8817	7868	7393	34597	

4.3 Distribution of type 1 wins with Black to move by Pawn's square

	A	B	C	D		Number of moves White needs to win
8	0	0	0	0	0	0
7	9	29	40	42	120	1
6	24	51	64	90	229	2
5	62	92	100	128	382	3
4	116	150	154	102	522	4
3	171	208	111	57	547	5
2	173	210	116	63	562	5
	555	740	585	482	2362	

4.4 Distribution of type 2 wins with Black to move by Pawn's square

	A	B	C	D	
8	3321	3218	3223	3223	12985
7	53	67	88	88	296
6	101	168	239	333	841
5	146	283	393	536	1358
4	217	531	723	727	2198
3	350	941	976	979	3246
2	372	1098	1185	1249	3904
	4560	6306	6827	7135	24828

5. The number of wins and draws at each depth

The main column of each of the following four tables is based on values given by Clarke (1975), although they have all been independently verified. A few minor printing errors in Clarke's paper are also corrected here.

Positions which are won for White are classified by the number of moves, N, needed for White to win, either with White to move initially or, in the case of Black to move initially, after Black's best move.

Assuming that White plays to minimize the depth from the nearest terminal position and Black plays to maximize it, the values given are the number of wins at depth 2N-1 and 2N ply, with White and Black to move respectively.

Positions which are drawn are similarly classified by the number of moves , N, taken by White to reach a terminal position, either with White to move initially or, in the case of Black to move, after Black's best move.

In this case, it is assumed that White plays to maximize the depth from the nearest terminal position and Black plays to minimize it.

The values given are thus the number of draws at depth 2N-1 and 2N ply, with White and Black to move, respectively. One further type of position, also tabulated, arises where the best play for both sides is a sequence of moves which eventually

repeats the original position. Such positions transform into one another in cycles of moves, indefinitely repeated.

5.1 Breakdown of won positions by number of moves required

WHITE TO MOVE

N	Number of positions in which White wins in N moves	Type 0	Type 1	Others
1	12750	12738	12	0
2	11300	10964	83	253
3	9624	8837	215	572
4	7864	6531	379	954
5	10511	8153	1033	1325
6	2564			
7	1416			
8	1457			
9	1122			
10	941			
11	685			
12	670			
13	586			
14	572			
15	307			
16	78			
17	22			
18	8			
19	3			
TOTAL	62480			

BLACK TO MOVE

N	*Number of positions in which White wins in N *moves	Type 0	Type 1	Type 2
0	12985*	-	-	-
1	11278	11158	120	0
2	9513	8994	229	290
3	7604	6646	382	576
4	5729	4279	522	928
5	5777	3520	1109	1148
6	1982			
7	1433			
8	1315			
9	966			
10	795			
11	597			
12	560			
13	502			
14	481			
15	213			
16	36			
17	15			
18	4			
19	2			
TOTAL	61787			

* Terminal positions

(Note that twelve of these have no legal antecedents in the set of positions under consideration, e.g. WK:C8 ,BK:A8, WP:B8.

The position could arise in play, however, as the result of a Pawn capture on B8).

5.2 Breakdown of drawn positions by number of moves required

WHITE TO MOVE

N	Number of positions in which White draws in N moves
1	2730
2	3243
3	2765
4	1405
5	504
6	130
Drawn by repetition	8405
TOTAL	19182

BLACK TO MOVE

N	Number of positions in which White draws in N moves
0	10109*
1	6156
2	5454
3	3172
4	1396
5	438
6	59
Drawn by repetition	9421
TOTAL	36205

* Terminal positions

6. The critical positions (White to move)

One particularly important set of positions with White to move consists of all positions where there is only one winning move, with 'trivial' cases excluded.

In this context, a reasonable definition of 'trivial cases' might be as follows:

(a) all positions in type 0 (where the Pawn can 'run')

(b) all positions in type 1 (where the Pawn can 'run')

(c) all positions where the White King is on the Pawn's file, somewhere ahead of the Pawn and Black is more than one rank or file outside the queening square of the Pawn (since then White can simply move the King off the Pawn's file and allow the Pawn to 'run').

Note that a Pawn on the second rank is considered to be on the third rank for purposes of constructing the queening square.

With such an exclusion the members of the set of 'critical positions' are all positions of type 2, as previously defined, and there are 1733 such positions in all.

6.1 Distribution of critical positions by Pawn's file

A	B	C	D	TOTAL
148	415	522	648	1733

7. The longest games

The longest possible game consists of a total of 38 ply and begins with Black to move in either of the positions

(a) WK:H3, BK:H5, WP:B2

(b) WK:H3, BK:H6, WP:B2

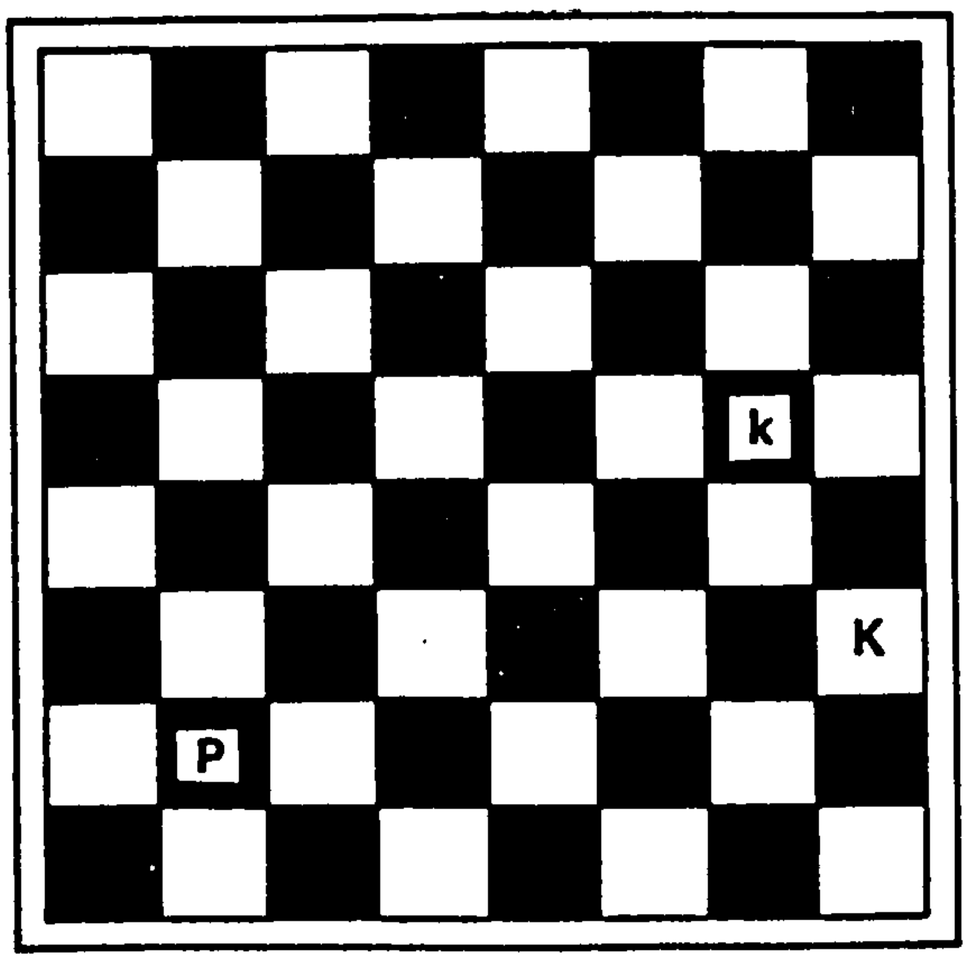
After Black's best move (Kg5 in both positions), the following position is reached, WK:H3, BK:G5, WP:B2

This is one of the three positions from which White needs 19 moves (37 ply) to win. The other two are

WK: H2, BK:G5, WP:B2

and WK:G2, BK:G5, WP:B2

In each case White's best move, 1.Kg3, transposes to the same position with Black to move.



One of the many possible'longest' variations is given below, with all possible deviations from the main line indicated.

1. ... Kf5

2. Kf3 Ke5

3. Ke3 Kd5

4. Kd3 Kc5

5. Kc3 Kb5

6. Kb3 Kc5

(6. ... Ka5, Ka6, Kc6 are equally good alternatives)

7. Ka4 Kb6

8. Kb4 Ka6

(8. ... Ka7, Kc6, Kc7 are equally good)

9. Kc5 Kb7

10. Kb5 Ka7
(not 10. ... Kc7)

11. Kc6 Ka8!

12. Kb6
(12.b4 is equally good)

12. ... Kb8

13. b4
(13.b3 is equally good)

13. ... Ka8
(Not 13. ... Kc8)

14. b5
(14. Ka6 is equally good)

14. ... Kb8

15. Ka6!
(Not Kc6)

15. ... Ka8
(15. ... Kc7 and Kc8 are equally good)

16. b6 Kb8

17. b7 Kc7

18. Ka7 any

19. b8 and wins.

References

Bramer, M.A. Forthcoming PhD. Thesis. 1977, The Open University.

Clarke, M.R.B. The Computational Complexity of King and Pawn Versus King. 1975, Mimeo. Forthcoming in Clarke, M.R.B. (ed.), Advances in Computer Chess 1. Edinburgh University Press.

King and Pawn against King:

using effective distance

M A BRAMER

March 1977

CONTENTS

Acknowledgements

I should like to thank Michael Clarke of Queen Mary College, London, for the use of his King and Pawn against King database, and Professor Mike Pengelly of the Mathematics Faculty, The Open University, for originally suggesting the idea of 'effective distance'.

1. Introduction

This report is concerned with the effective distance between two squares on a (finite or infinite) chessboard, that is, the shortest distance (measured in single-step King moves) between the two squares, allowing for any necessary detour around an inaccessible region of the board, in particular a nine-square region centred on a given square.

Formulae are given for both an infinite board and a board which is infinite except for a fixed edge. The formulae are then used as the basis for an implementation in a computer subroutine of a predicate function applicable to the King and Pawn against King endgame: 'Pawn can run', i.e. the player with the Pawn can advance it and continue to do so until it reaches its promotion square on the eighth rank. The subroutine has been verified empirically using a database of information on King and Pawn against King and correctly classifies all positions with Black to move (where White is the side with the Pawn). The database is described further in Clarke (1975).

This report is presented both as a contribution to the several research projects currently in progress in the task area of Chess endgames (see, for example, Bramer (1977))and in the hope of stimulating interest in the mathematical problems associated with the geometry of the chessboard and the equivalent (finite or infinite) two-dimensional lattices.

In Section 2 the role of effective distance in implementing the 'Pawn can run' predicate function for King and Pawn against King is shown. In Section 3, formulae are derived for effective distance for the general case of a board which is infinite in all directions and the case of a board which is infinite except for a single fixed edge.

In the case of the 'Pawn can run' predicate for King and Pawn against King on a finite 8 by 8 board, a number of simplifications to the general formulae are possible and these are discussed in Section 4. A number of other considerations specific to King and Pawn against King, in particular the concept of effective queening square, are introduced here. A listing of a FORTRAN subroutine implementing 'Pawn can run' is given in Section 5.

1.1 Notation and conventions

In the King and Pawn against King diagrams which follow, White is assumed to be playing up the board in all cases and in each position it is Black to move. The White King, Black King and White Pawn are denoted by K, k and P, respectively.

It is assumed that positions where Black to move is stalemated are excluded from consideration and that the Pawn stands on ranks 2 to 7, inclusive. The horizontal and vertical co-ordinates of the White King (relative to White's bottom left-hand corner of the board) are denoted by WK1 and WK2, respectively. Similarly the co-ordinates of the Black King and the White Pawn are represented by BK1 and BK2 for the former and WP1 and WP2 for the latter. In this notation the position White King: QR2, Black King: QR5 (White's QR4), White Pawn: QR3 would be represented by

values of 1, 2, 1, 4, 1, 3 for WK1, WK2, BK1, BK2, WP1 and WP2, respectively. WP3 is used to denote the 'effective rank' of the Pawn. It has the same value as WP2, except when WP2 = 2, in which case WP3 is 3.

This value is used when calculating the number of moves needed for the Pawn to reach the eighth rank, to allow for an initial double move from the second rank (WP2 = 2).

2. Using effective distance for King and Pawn against King

2.1 Definitions and the'rule of the square'

In a position where the effect of the White King can be ignored, the shortest distance between the Black King and the Pawn's promotion square, measured in single-step King moves, is simply

$$\max\{\text{abs}(BK1-WP1),\ 8-BK2\}$$

This value will be called the block distance between the two squares. If it is greater than 9-WP3 (ie. one more than the number of moves taken by the Pawn to reach the promotion square - to allow for Black moving first), then the Pawn can run. This is, in fact, a reformulation of the well-known 'rule of the square'.

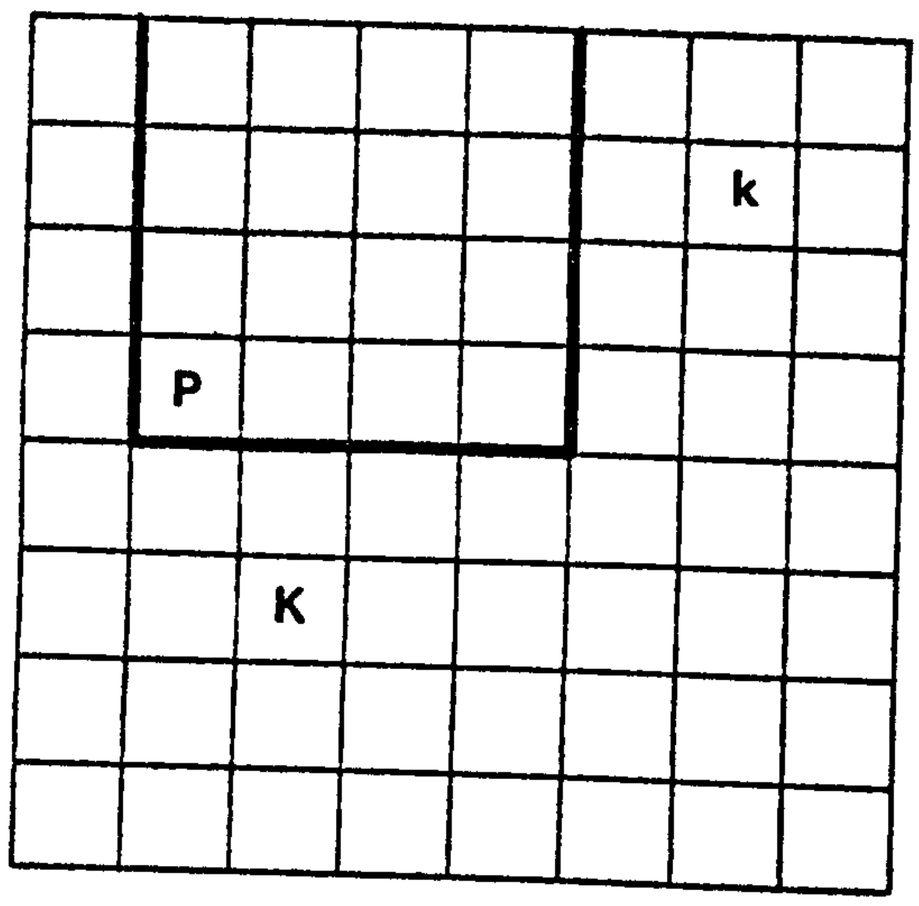


Figure 1 (Black to move)

If Black cannot enter the 'queening square' (as in Figure 1), then he loses, assuming of course that White's own King does not block the Pawn.

The effective distance between two squares is defined in general as the smallest number of King moves needed to travel from one square to the other allowing for detours around fixed 'holes' of inaccessible squares on the board.

In the case of King and Pawn against King it is useful to consider a hypothetical hole of (up to) nine squares centred on the White King. These correspond, of course, to the squares the Black King may not legally occupy, adjacent to (or on the same square as) the White King.

Provided the effective distance between the Black King and the Pawn's promotion square is greater than 9-WP3, the Pawn can run (assuming that it is not obstructed by the White King and subject to certain exceptions discussed in Section 4). This is, in fact, a generalized form of the 'rule of the square'.

For convenience, the effective distance between any two squares can be written as

effective distance = block distance + incval

where block distance is the distance on a board without 'holes' and incval is a non-negative distance increment. Incval can be calculated using the values of two variables intbk and intqsq, where

intbk = abs (WK1-BK1) + BK2-WK2
and intqsq = 8 - abs (WK1-WP1) - WK2

These are, in fact, the values of the intercepts on the White King's file of diagonal lines drawn through the Black King and the Pawn's promotion square, respectively (taking the White King to be at intercept zero). In general, incval is non-zero when both intbk and intqsq lie in the range -2 to +2 inclusive and the White King is closer to the promotion square than the Black.

2.2 Examples

Figures 2-9 below show some of the possible cases for King and Pawn against King.

The nine-square 'hole' around the White King is marked in each case, together with the diagonal lines through the Black King and the Pawn's promotion square.

From the examples given, it will be seen that incval is closely related to the smallest number of squares in the inaccessible region around the White King which lie on any of the Black King's shortest paths to the promotion square (in the block distance calculation).

Figure 9 is an example of a special case for which an adjustment to the basic formula for incval is needed to allow for the effect of the upper edge of the board, since Black's quickest route to the promotion square would involve moves off the board, to the "ninth" rank.

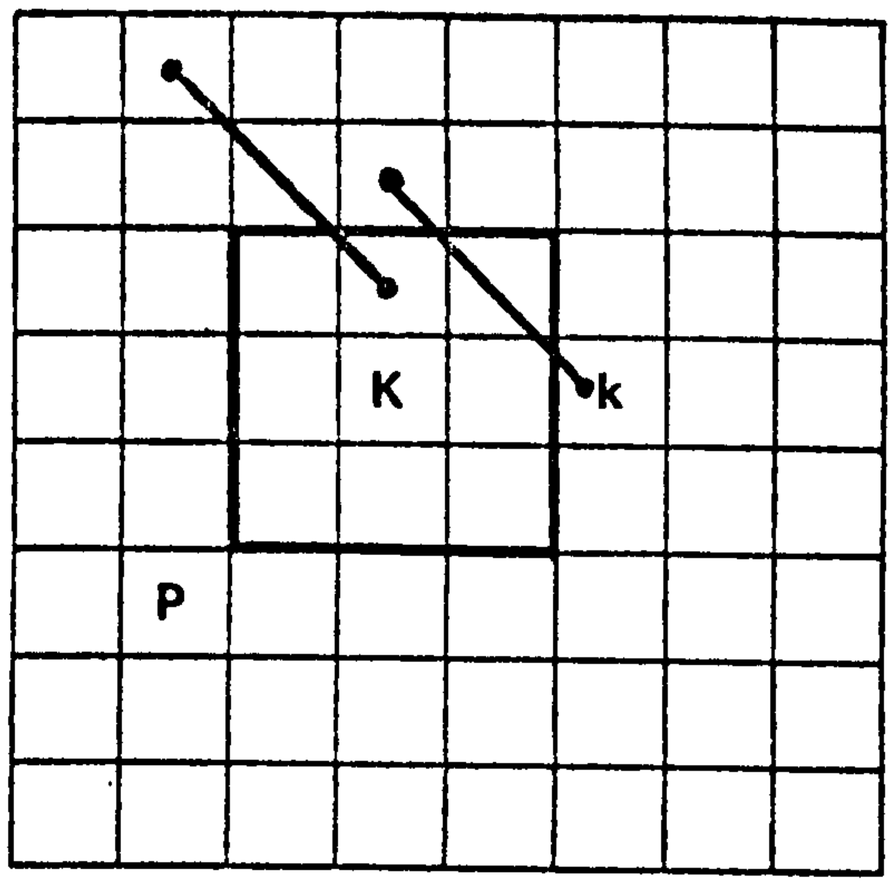


Figure 2

block distance = 4
incval = 1
intbk = 2
intqsq = 1

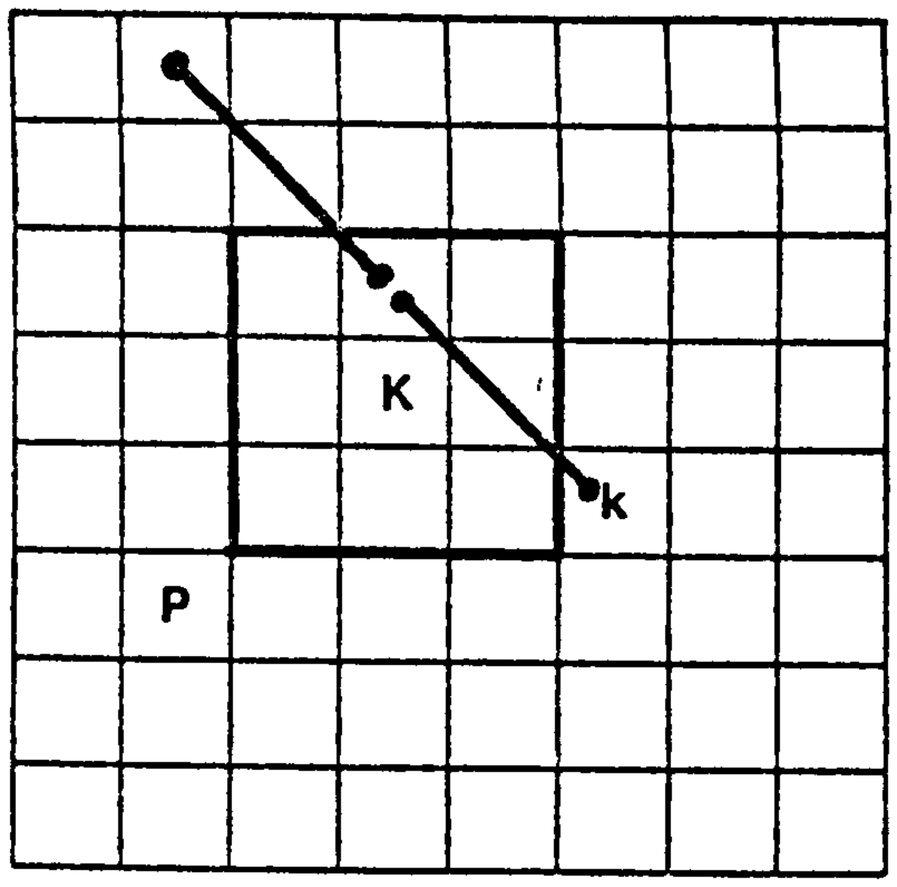


Figure 3

block distance = 4
incval = 2
intbk = 1
intqsq = 1

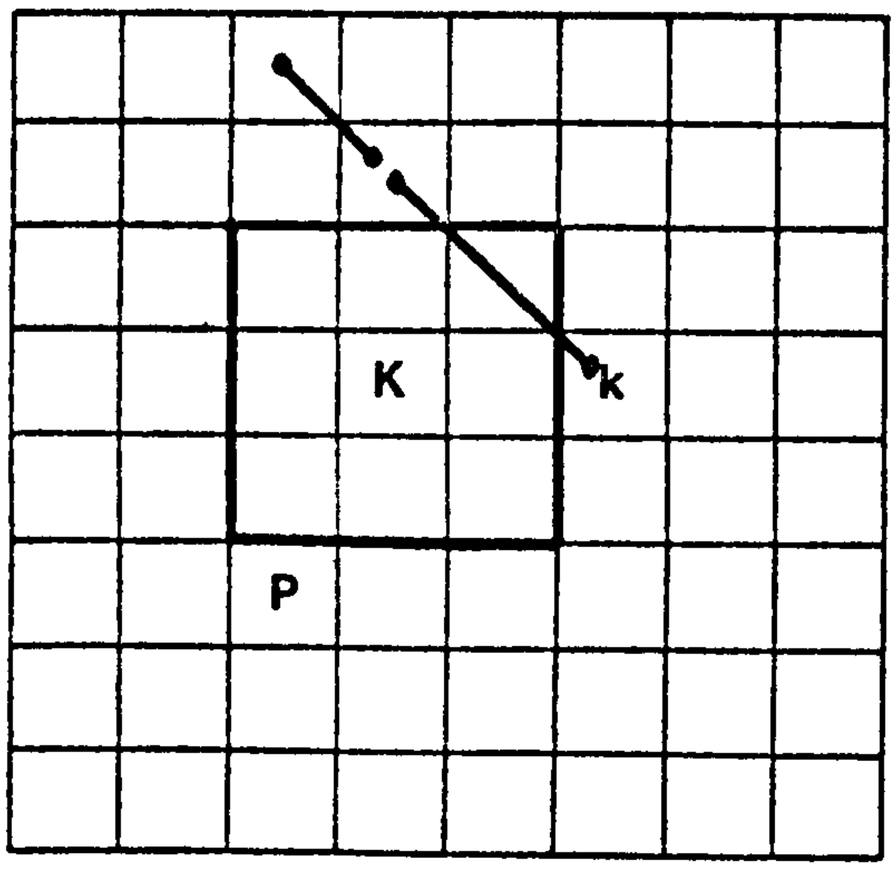


Figure 4

block distance = 3
incval = 1
intbk = 2
intqsq = 2

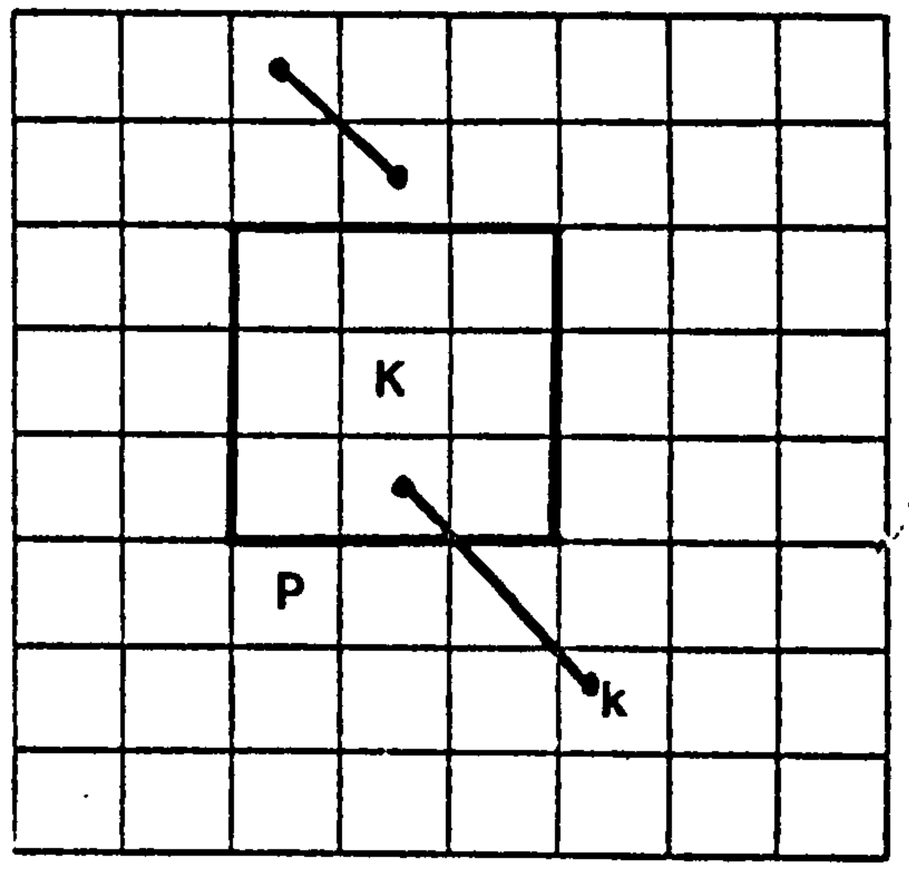


Figure 5

block distance = 6
incval = 1
intbk = -1
intqsq = 2

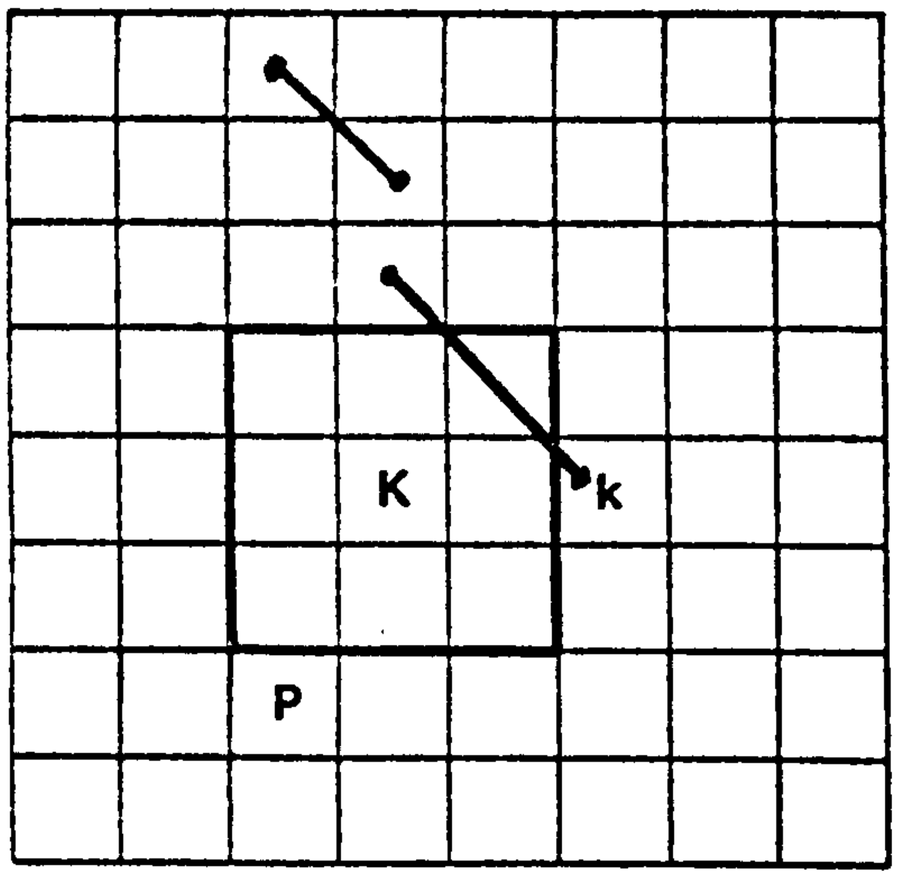


Figure 6

block distance = 4
incval = 0
intbk = 2
intqsq = 3

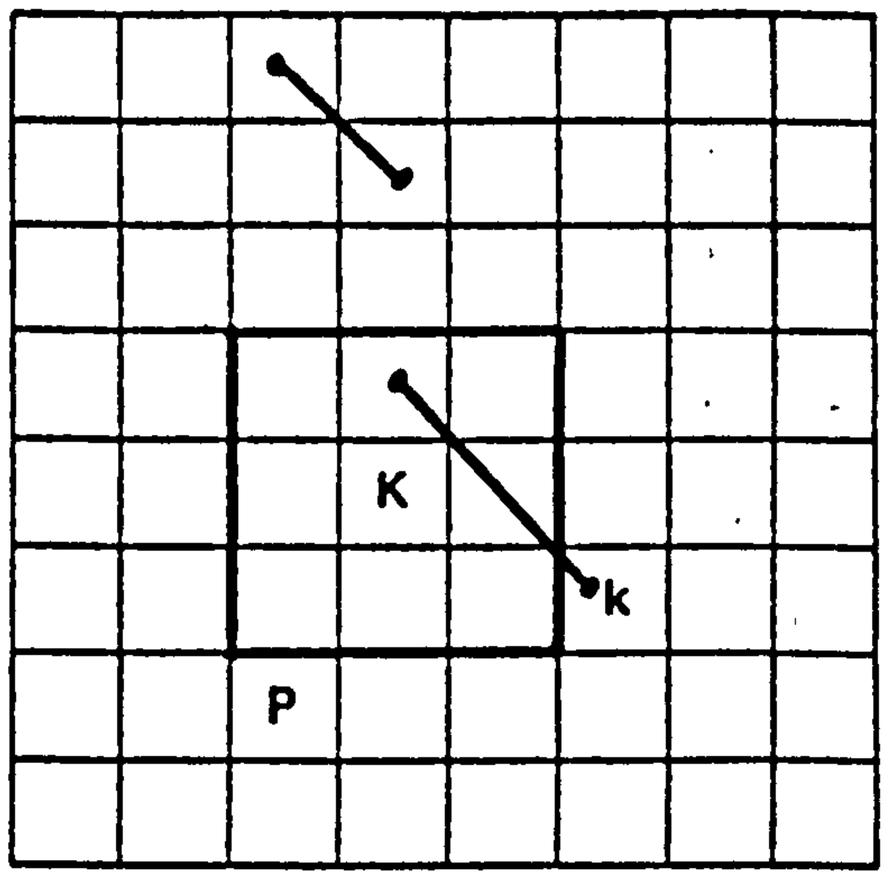


Figure 7

block distance = 5
incval = 0
intbk = 1
intqsq = 3

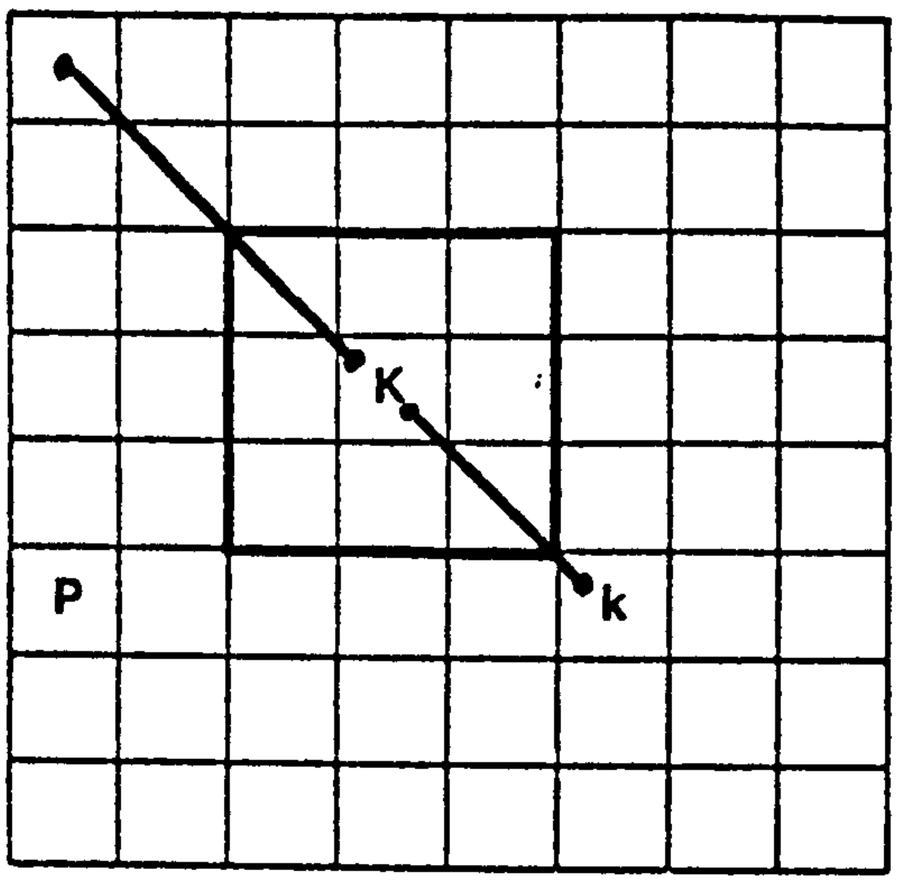


Figure 8

block distance = 5
incval = 3
intbk = 0
intqsq = 0

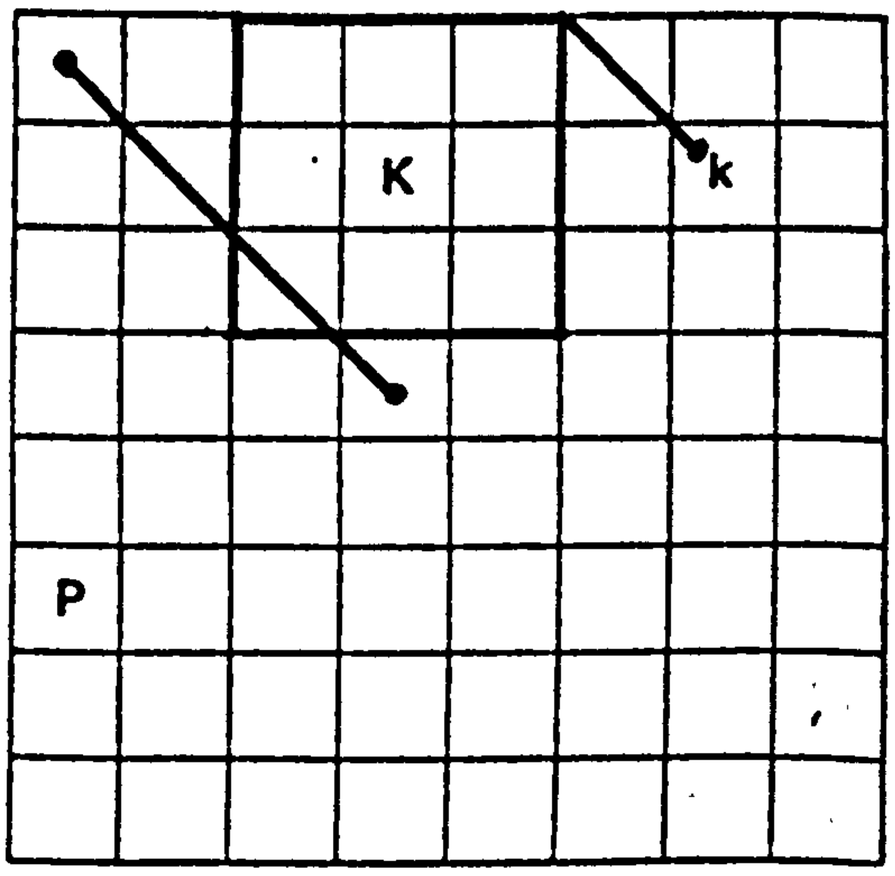


Figure 9

block distance = 6
incval = 1
intbk = 3
intqsq = -2

3. Calculating effective distance on an infinite board

The general problem will be considered of calculating the effective distance (measured in single-step King moves) between two squares A and B on an infinite chessboard, subject to the presence of an inaccessible region of nine squares centred on square K. The horizontal and vertical co-ordinates of A, B and K will be denoted by (a_1,a_2), (b_1,b_2) and (k_1,k_2), respectively. It will be assumed that A, B and K are three distinct squares. If A or B is inside the inaccessible region around K, the effective distance between A and B will be considered infinite and such positions are excluded from the analysis which follows.

As in Section 2, the effective distance between A and B will be written in the form

block distance + incval

where 'block distance' is the distance on a board with no inaccessible region, ie. $\max\{\text{abs}\,(a_1-b_1), \text{abs}\,(a_2-b_2)\}$ and incval is a non-negative distance increment. Formulae for effective distance will be given in terms of expressions for incval.

In the remainder of this section it will be assumed without loss of generality that $k_1 \geqslant a_1$ and $k_2 \leqslant a_2$, ie. that K is inside or on the edge of the quadrant which lies below and to the right of A.

3.1 The case of a fully infinite board

In the case of a board which is infinite in all directions, incval is zero unless K is at least as close to square A as B is, measured in both the horizontal and the vertical directions. That is, the conditions

(i) $b_1 \geqslant k_1$ if $k_1 > a_1$

and (ii) $b_2 \leqslant k_2$ if $k_2 < a_2$

must both be satisfied for a non-zero value of incval.

It is convenient to assume that B satisfies the additional conditions $b_1 \geqslant k_1$ when $k_1 = a_1$ and $b_2 \leqslant k_2$ when $k_2 = a_2$, again without loss of generality.

When these conditions are satisfied, the value of incval can most easily be calculated in terms of the intercepts on the file or rank through K of the diagonals through A and B. The intercepts on the file through K (the vertical intercepts) of the diagonals through A and B will be denoted by inta and intb, respectively.

These are, in fact, the same both in magnitude and in sign as the corresponding horizontal intercepts on the rank through K, with the standard orientation ($b_1 \geqslant k_1 \geqslant a_1$ and $b_2 \leqslant k_2 \leqslant a_2$) chosen. Horizontal and vertical intercepts will not therefore usually need to be distinguished in what follows.

In Figure 10 the intercepts of A and B are +1 and +2, respectively, where K itself is at intercept zero.

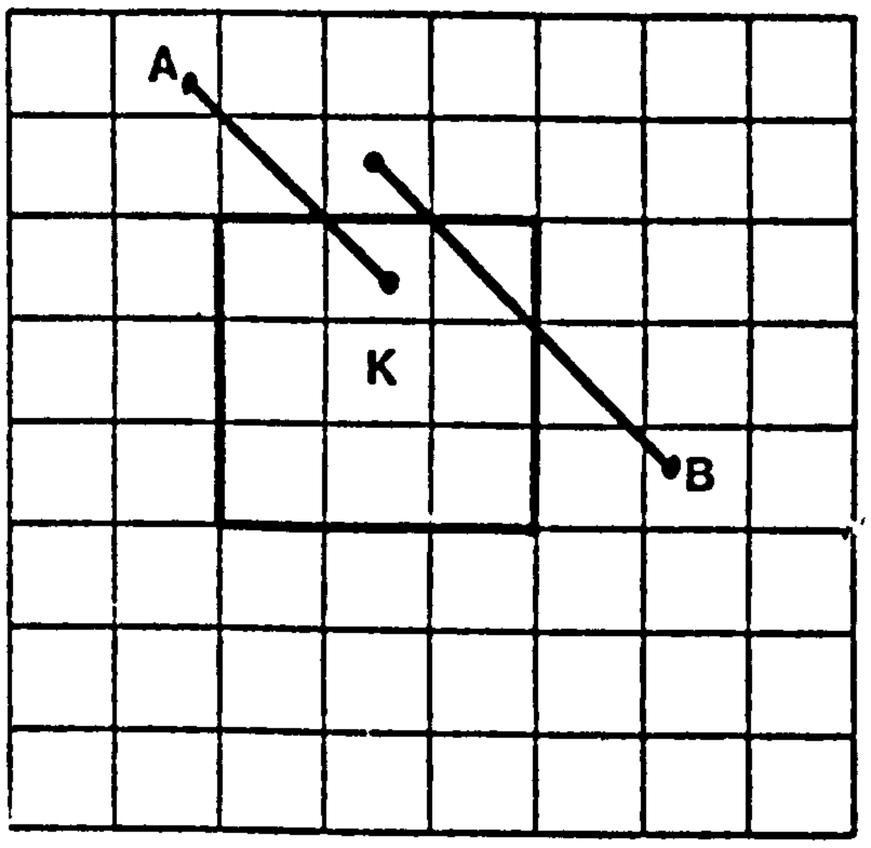


Figure 10. (The board is assumed to continue infinitely in all directions)

In general, $\text{inta} = (a_1 + a_2) - (k_1 + k_2)$

$\text{intb} = (b_1 + b_2) - (k_1 + k_2)$

With one important exception, discussed subsequently, incval is only non-zero when both inta and intb are in the range -2 to +2, inclusive, i.e. both A and B are no more than 2 "diagonal ranks" away from K.

The following table shows the value of incval corresponding to each combination of values of inta and intb.

abs (inta)	abs (intb)	incval
0	0	3
0	1	2
1	0	2
1	1	2
0	2	1
2	0	1
1	2	1
2	1	1
2	2	1
(Other values)		0

Thus in figure 10, inta = 1 and intb = 2 so incval = 1, which is correct since the effective distance from A to B is 6, whereas the block distance is only 5.

The one exception mentioned previously occurs when either A or B is two squares from K on the same rank or file. With the standard orientation of A, B and K adopted in this section, there are two positions of A and two of B for a given position of K where this condition is satisfied, of which Figure 11 is a typical example.

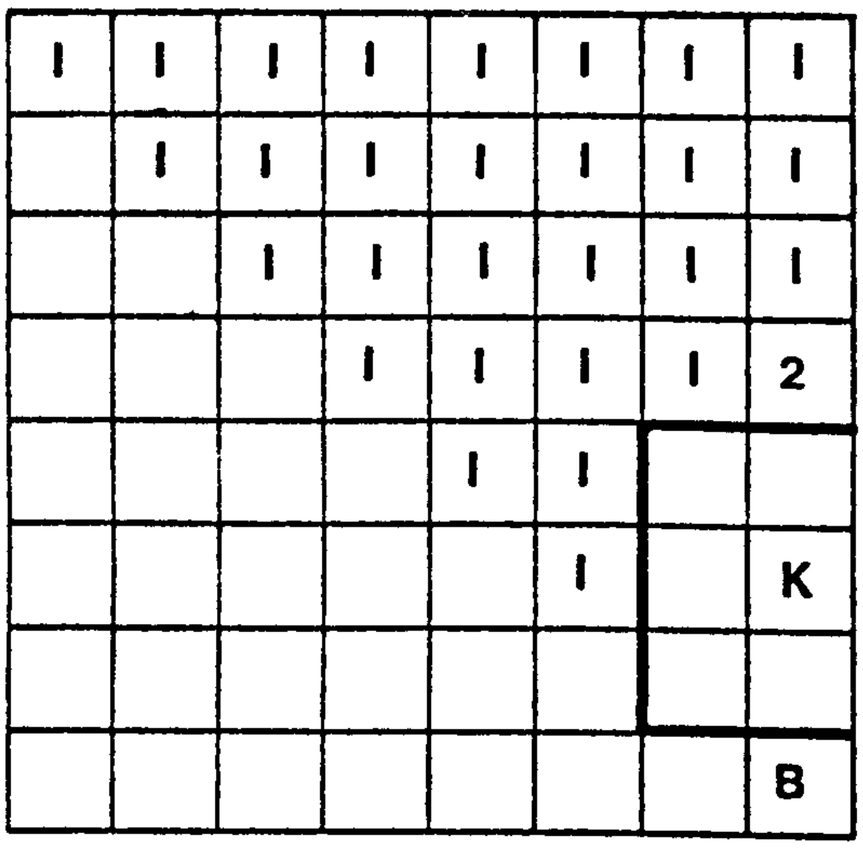


Figure 11 (The board is assumed to continue infinitely in all directions)

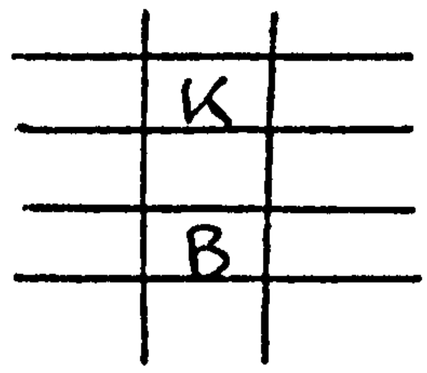
The number in each square shows the value of incval associated with the effective distance from B to that square. Only non-zero values are shown. A must lie to the left of or on the same file as K, so all incval values are zero for files to the right of that file.

The four possible cases are:

(i) $k_1 = b_1$ and $k_2 = b_2 + 2$

incval = 2 if $a_1 = k_1$ and $a_2 = k_2 + 2$

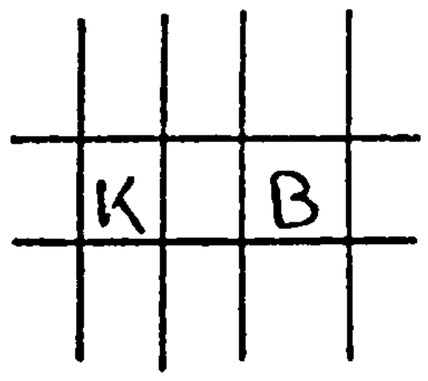
incval = 1 if inta $\geq$ -2 otherwise



(ii) $k_1 = b_1 - 2$ and $k_2 = b_2$

incval = 2 if $a_1 = k_1 - 2$ and $a_2 = k_2$

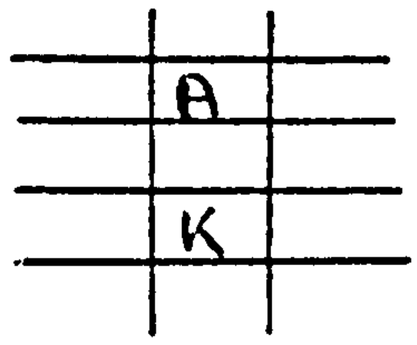
incval = 1 if inta ≤ 2 otherwise



(iii) $k_1 = a_1$ and $k_2 = a_2 - 2$

incval = 2 if $b_1 = k_1$ and $b_2 = k_2 - 2$

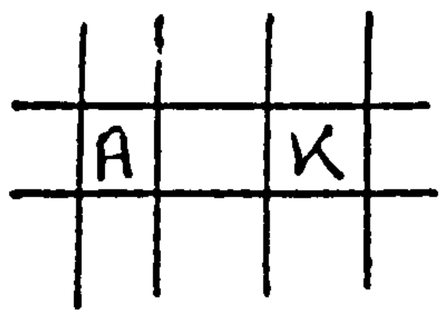
incval = 1 if intb ≤ 2 otherwise



(iv) $k_1 = a_1 + 2$ and $k_2 = a_2$

incval = 2 if $b_1 = k_1 + 2$ and $b_2 = k_2$

incval = 1 if intb ≥ -2 otherwise

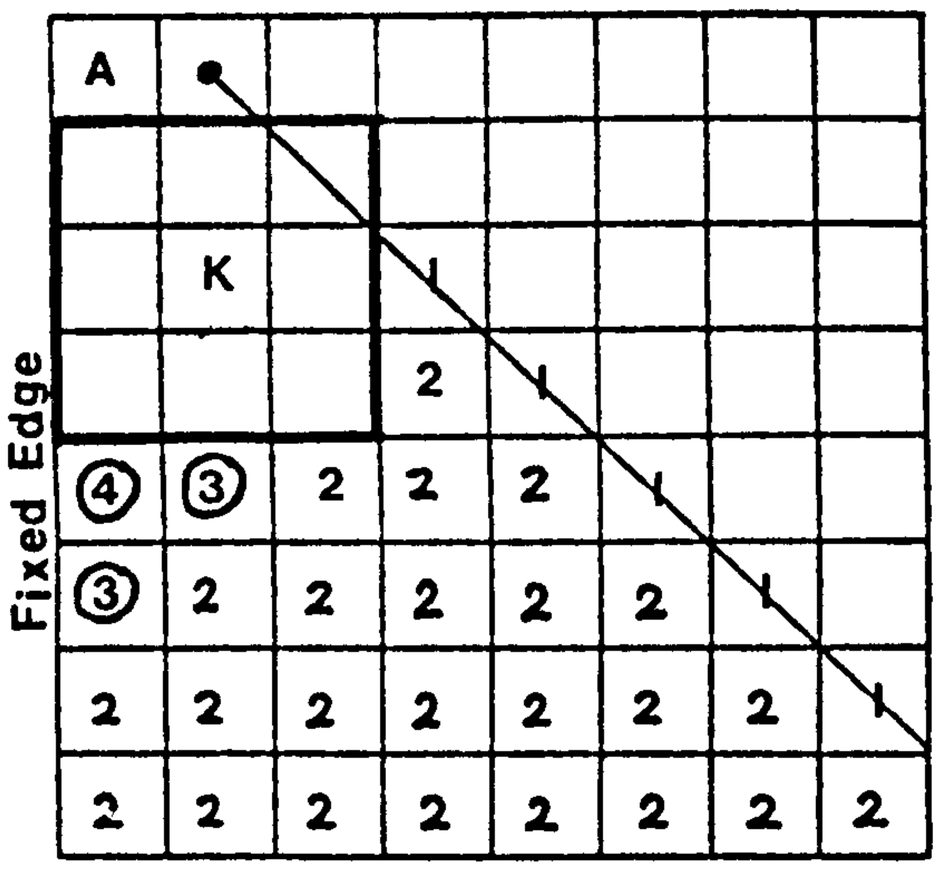


3.2 An infinite board with a fixed edge

An adjustment to the formulae given in Section 3.1 needs to be made in certain cases when either A or B lies on a vertical (or horizontal) fixed edge of the board, with K on the adjacent file (or rank), since the shortest path between A and B would then involve moves off the edge of the board.

Allowing more than one fixed edge produces no additional complications.

Taking the example of A on the left-hand edge, with K on the adjacent file, only two cases need to be distinguished: those where K is either 2 or 3 ranks below A, as shown in Figures 12 and 13.


A
K
Fixed Edge
④
③
2
③
2
2
2

It will be seen that in this case the restrictions on the position of B which were necessary in Section 3.1 for non-zero incval do not fully apply and B can also be on the same edge as A.

With the relative orientation of A and K adopted in this section, positions equivalent to Figures 12 and 13 can only arise with A on a top or a left-hand edge. B can either be on the same edge as A or on a bottom or right-hand edge (in which case A could not be on the same edge because of the orientation adopted).

In each case there are 3 exceptional positions of B (or A) superimposed on a simple general pattern, thus in Figure 12

$$\text{incval} = \begin{cases} 1 \text{ if intb} = 2 \\ 2 \text{ if intb} < 2 \end{cases}$$

and in Figure 13

$$\text{incval} = 1 \text{ if intb} \leq 2$$

with the exception of the three ringed squares in each case.

All other board edge positions are handled correctly by the formulae given in Section 3.1.

4. Implementing 'Pawn can run' for King and Pawn against King

The subroutine given in Section 5 makes use of the effective distance formulae developed in Section 3 in a complete implementation of the 'Pawn can run' predicate for King and Pawn against King.

A number of additional considerations specific to the nature of the King and Pawn against King endgame have been incorporated in the subroutine, including the idea of effective queening square, and these are described in Sections 4.1 to 4.3 below.

Positions where either King is ahead of the Pawn on the same file are excluded immediately by the subroutine since the Pawn certainly cannot 'run' in such cases. Similarly, in positions where the Black King is more than one rank or file outside the square of the Pawn (and the White King does not obstruct the Pawn), 'Pawn can run' is taken as true, without any calculation of effective distance being made (since it must be at least as great as the block distance of the Black King from the promotion square).

In principle the formulae derived in Section 3 for the case of an infinite board with a single fixed edge can be applied directly to the present case of a finite 8 by 8 board, since it is never necessary to consider the effect of more than one edge of the board in any given position.

In practice however, a number of simplifications can be made to the general formulae. For example, since it is known that the Pawn's promotion square is always on the eighth rank, there is no need to allow for the effect of a fixed edge at the bottom of the board.

In the case of the adjustment to the general formula which theoretically is needed whenever the Black King (or the promotion square) is two squares away from the White King, on the same rank or file, it turns out that it is only necessary to make this refinement when the Black King is two ranks vertically below the White, or when the White King is on the seventh or eighth rank.

The further adjustments to allow for the effect of a fixed edge of the board (given in Section 3.2) are only necessary in practice when the Black King is on a vertical edge of the board or when the White King is on the seventh rank. In the latter case it can easily be shown that positive values of intqsq can never occur. Further simplifications are also possible because it is known that the Pawn takes at most 5 moves to reach the promotion square and thus all effective distances greater than 6 are equivalent for the purposes of the subroutine. It is therefore not necessary fully to implement the calculation of the values of incval ringed in Figures 12-13 in Section 3.2 and a number of ad hoc simplifications have been made in the subroutine in the interests of computational efficiency.

4.1 Effective queening square

In positions where the White King is on the file adjacent to the Pawn and on the seventh or eighth rank (whether on the same side of the Pawn as the Black King or not), the effective distance of the Black King from the promotion square can be considered to be infinite. However, it is not true that 'Pawn can run' in all such positions. In this case it is helpful to consider the concept of effective queening square, ie. the square on the Pawn's file two ranks below the White King. Provided the Pawn can reach this square without being immediately captured, it can go on to reach the eighth rank and promote.

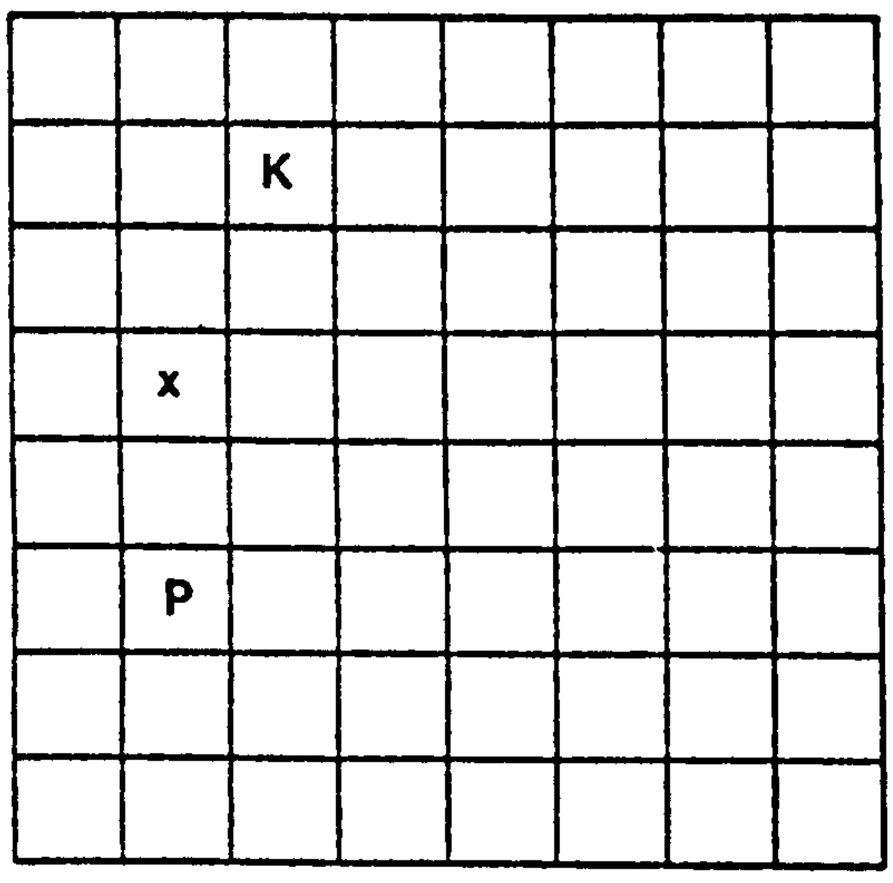


Figure 14

Thus in Figure 14, the Pawn's effective queening square is marked with a cross.

The effective distance formulae given in Section 3 can also be used to calculate the number of moves needed by the Black King to reach the effective queening square and, in this case, the fixed relative position of the White King allows considerable simplifications to be made.

Figures 15 and 16 show the only non-zero values of incval for a White King on the seventh or eighth rank respectively.
In each case, the Black King must be on the same side of the Pawn as the White King.

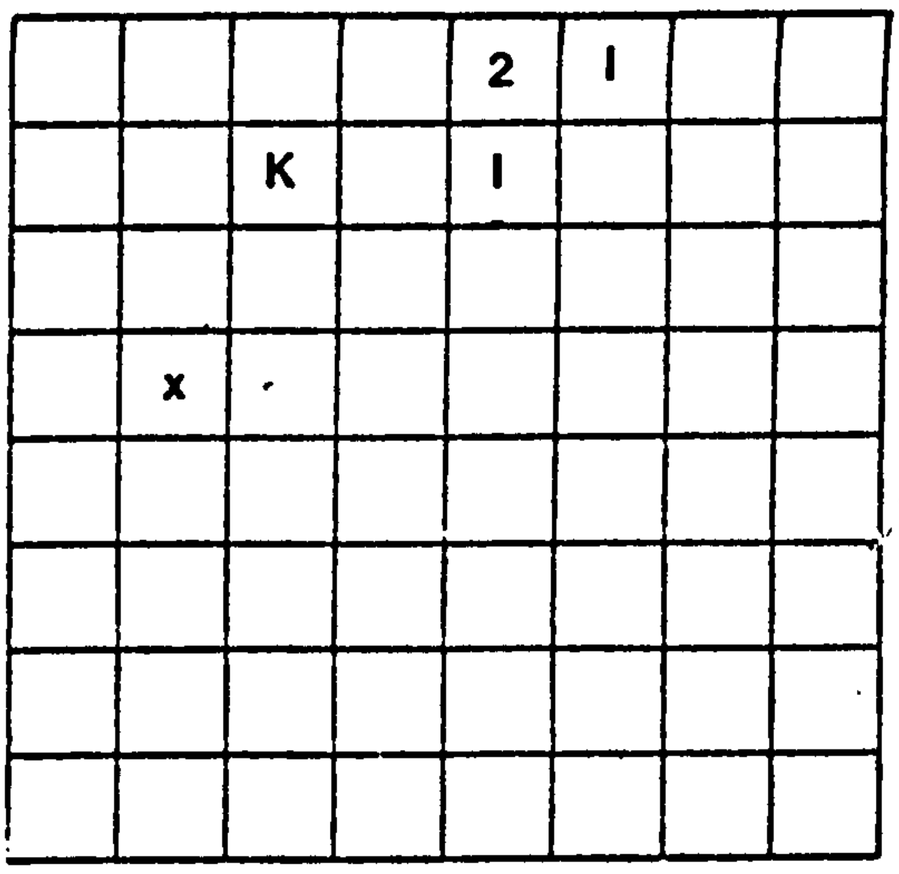


Figure 15

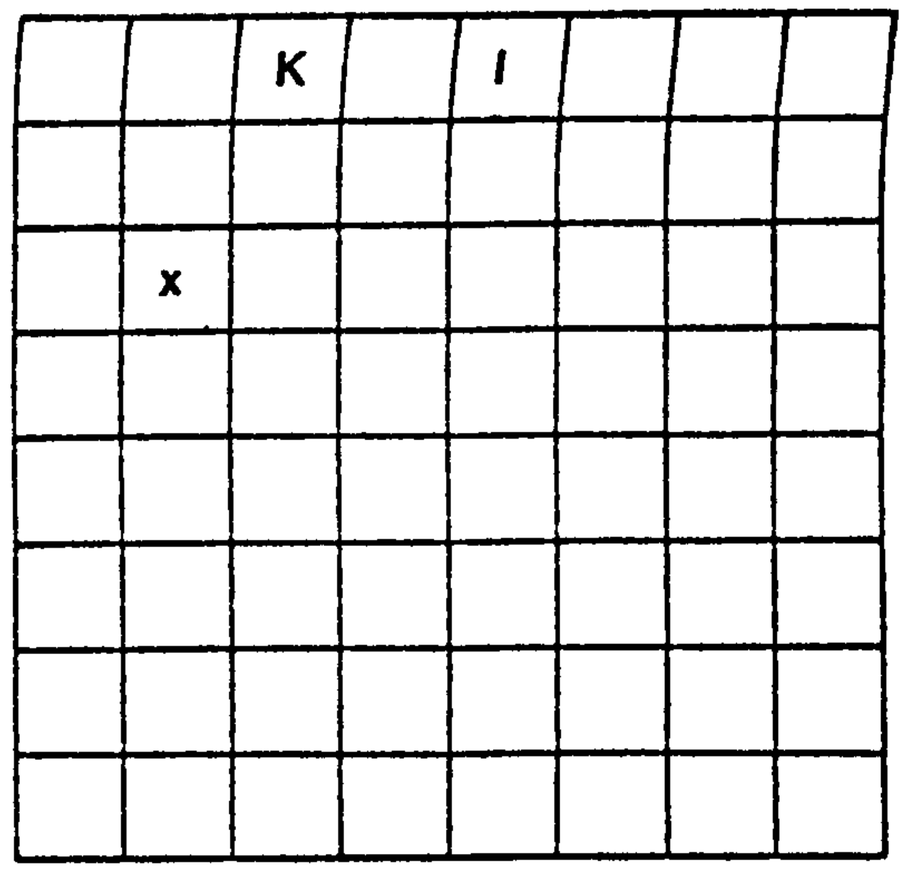


Figure 16

4.2 Special cases: Allowing for the checking power of the Pawn

In the preceding analysis, the checking power of the Pawn has been entirely ignored. However, it plays an important part in two types of position.

4.2.1. White King on the sixth rank

In certain positions it is necessary to increase the value of incval (and thus the effective distance of the Black King from the promotion square) by one, to allow for the possibility of the Pawn itself preventing Black's King from reaching the queening square. Thus in Figure 17, play may continue:

1... Kg7
2.c5 Kf8
3.c6 Ke8
4.c7

and now Black's move Kd8 is prevented by the Pawn and so White queens next move.

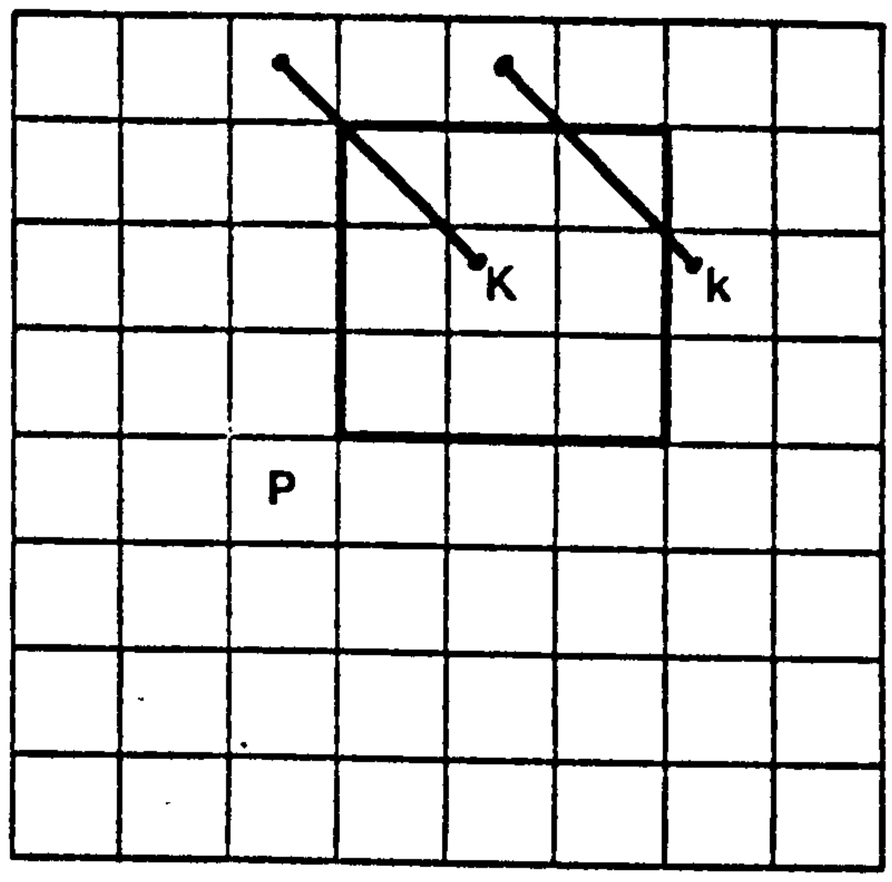


Figure 17

block distance = 4
incval = 2
intbk = 2
intqsq = 0

The value of incval in Figure 17 should be counted as 2 not 1. Incval should be increased by one whenever the White King is on the sixth rank, no more than two files away from the Pawn, and the Black King's shortest route to the promotion square involves a move to the eighth rank, two files from the Pawn (on the same side as the White King) immediately before the advancing Pawn reaches the seventh rank.

The full implementation of this condition is fairly complex, but an adequate approximation (used in the subroutine) is as follows:

WK2 = 6 AND intbk $\geqslant$0 AND abs (WK1 - WP1) $\leqslant$2 AND abs (BK1 - WP1) $>$1.

4.2.2 Other special cases

Figures 18-20 below show 3 types of position in which the Pawn can run solely because of its checking power. Thus for example in Figure 20, Black could otherwise draw by Kc3.

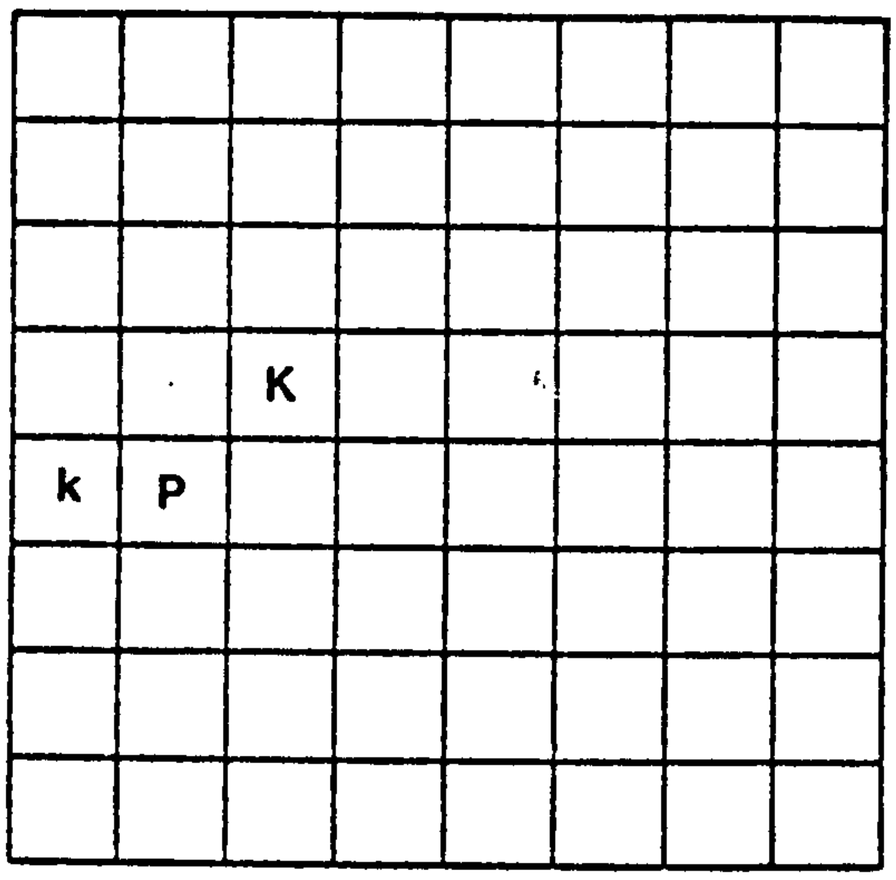


Figure 18

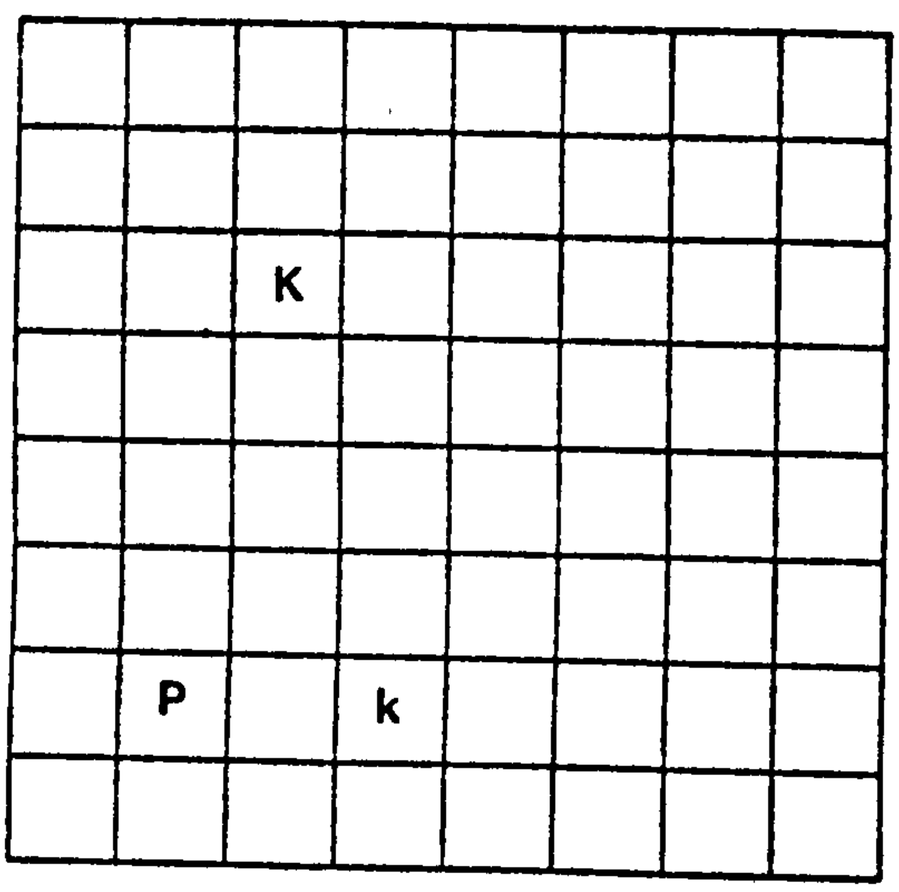


Figure 19

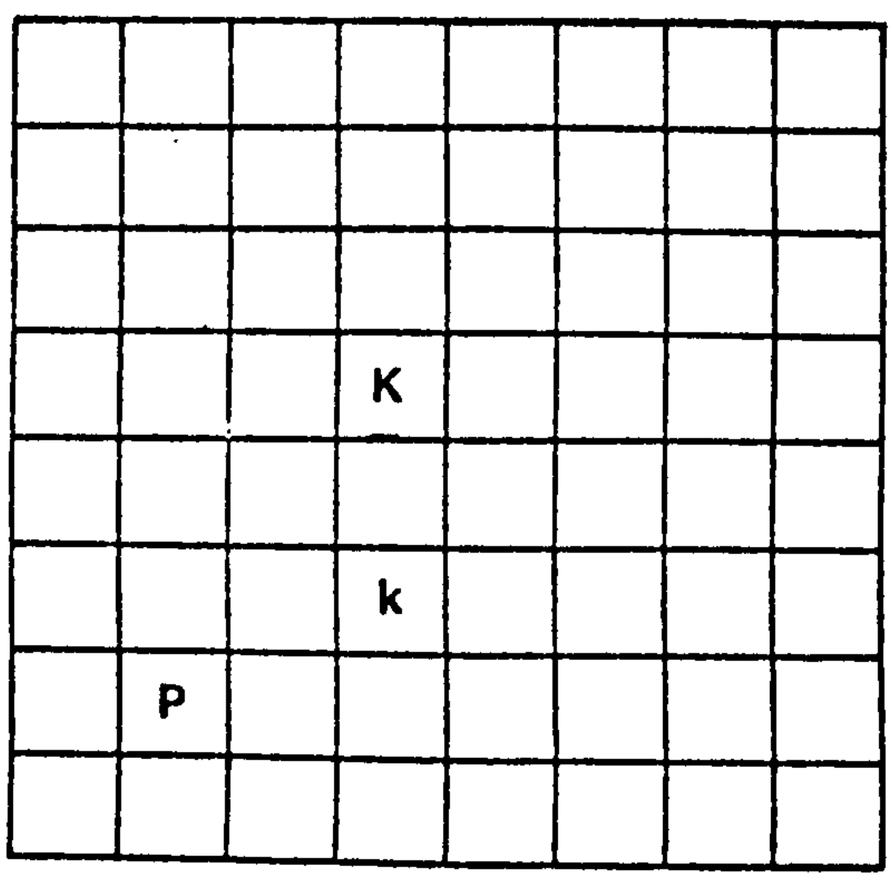


Figure 20

The full specification of the three types of position is as follows:

(i) (BK1 = 1 OR BK1 = 8) AND abs (BK1 - WP1) = 1
AND abs (BK1 - WK1) = 2 AND WK2 $\geqslant$ WP2
AND WP2 = BK2

(ii) WP2 = 2 AND BK2 = 2 AND sameside
AND abs (BK1 - WP1) = 2
AND {(abs (WK1 - WP1) = 1 AND WK2 = 6)
OR (WK1 = BK1 AND WK2 = 5)
OR (abs (WK1 - WP1) = 3 AND WK2 = 4)}

(iii) WP2 = 2 AND BK2 = 3 AND WK2 = 5
AND WK1 = BK1 AND abs (WK1 - WP1) = 2

where sameside is true if both Kings are on the same side of the Pawn. It is not necessary in practice to distinguish any other types of position where the Pawn's checking power is significant (if any exist).

4.3 Special cases: allowing for stalemate possibilities

There are four positions which need to be treated separately where the Pawn cannot 'run' because of stalemate considerations, although all other criteria are satisfied.

WK	BK	WP
C7	A8	B4
C7	A7	B5
F7	H8	G4
F7	H7	G5

Thus, for example, in the first position play might continue

1.....Ka7
2.Kb5 (giving the second positon)
2.....Ka8

and now 3.b6 gives stalemate.

5. Listing of the 'Pawn can run' subroutine

A listing of the subroutine, which is written in FORTRAN IV, is given below.

Variable IPOS is used to 'pack' the six co-ordinates which specify each position, giving a simple way of testing for individual positions.

Variable NSQ holds the rank of the 'effective queening square' of the Pawn.

The remainder of the subroutine is intended to be self-explanatory in the light of the preceding discussion.

```
      SUBROUTINE CRUN(WK1,WK2,BK1,BK2,WP1,WP2,IRES)
      INTEGER WK1,WK2,BK1,BK2,WP1,WP2,WP3
      LOGICAL NEARER,SAMESD
C     PAWN CAN RUN POSITIONS KPK BLACK TO MOVE
C     IT IS ASSUMED THAT  STALEMATE POSITIONS AND
C     TERMINAL POSITIONS WITH PAWN ON THE EIGHTH RANK
C     HAVE PREVIOUSLY BEEN TESTED FOR SEPARATELY.
C
C     THE PAWN IS ASSUMED TO BE ON RANKS 2-7 ONLY
C     ------------------------------------------
C
       IPOS= 100000*WK1+10000*WK2+1000*BK1+100*BK2+10*WP1+WP2
       WP3=WP2
       IF (WP2.EQ.2) WP3=3
       NEARER= (WK1.GT.WP1 .AND. BK1.GE.WK1)
      +    .OR. (WK1.LT.WP1 .AND. BK1.LE.WK1)
      +    .OR. (WK1.EQ.WP1)
       SAMESD= (WK1.GT.WP1 .AND. BK1.GT.WP1)
      +    .OR. (WK1.LT.WP1 .AND. BK1.LT.WP1)
C
C      SPECIAL CASES- EXCLUSIONS
C      -------------------------
C
       IF (IPOS.EQ.371824) GOTO 200
       IF (IPOS.EQ.371725) GOTO 200
       IF (IPOS.EQ.678874) GOTO 200
       IF (IPOS.EQ.678775) GOTO 200
C
C      EXCLUDE POSITIONS WHERE THE PAWN IS OBSTRUCTED
C      ----------------------------------------------
C
       IF (WK1.EQ.WP1 .AND. WK2.GT.WP2) GOTO 200
       IF (BK1.EQ.WP1 .AND. BK2.GT.WP2) GOTO 200
C
C      BK OUTSIDE THE SQUARE AND CANNOT ENTER IN ONE MOVE
C      --------------------------------------------------
C
       IF (MAXO(IABS(BK1-WP1),8-BK2).GT.9-WP3) GOTO 110
```

```
C
C       TEST FOR WK ADJACENT TO QUEENING SQUARE-  IF SO, USE
C       ------------------------------------------------------
C       EFFECTIVE QUEENING SQUARE (INCLUDING EFFECTIVE DISTANCE)
C       ------------------------------------------------------
C
        IF (IABS(WK1-WP1).NE.1 .OR. WK2.LT.7) GOTO 350
        NSQ=WK2-2
        IF (WP2.GT.NSQ) GOTO 110
        INCVAL=0
        IF (.NOT. SAMESD) GOTO 233
        IF (WK2.EQ.BK2 .AND. IABS(WK1-BK1).EQ.2) INCVAL=1
        IF (WK2.EQ.7 .AND. BK2.EQ.8 .AND.
       +  IABS(WK1-BK1).EQ.3) INCVAL=1
        IF (WK2.EQ.7 .AND. BK2.EQ.8 .AND.
       +  IABS(WK1-BK1) .EQ.2) INCVAL=2
      233 IF (MAX0(IABS(BK1-WP1),IABS(BK2-NSQ))
       +     +INCVAL .GT. NSQ+1-WP3) GOTO 110
        GOTO 250
C
C       EFFECTIVE DISTANCE OF BK FROM QUEENING SQUARE
C       ---------------------------------------------
C       - CALCULATE INCREMENT
C       ---------------------
C
      350 INCVAL=0
        INTQSQ=8-IABS(WK1-WP1)-WK2
        INTBK=IABS(WK1-BK1)+BK2-WK2
        IF (((BK1.EQ.1 .AND. WK1.EQ.2) .OR. (BK1.EQ.8 .AND. WK1.EQ.7))
       +  .AND. WK2.EQ.BK2+2) GOTO 288
        IF (.NOT. NEARER) GOTO 290
        IF (WK2.EQ.7 .AND. IABS(WK1-WP1).LE.3) GOTO 340
        IF (WK2.EQ.7 .AND. BK2.EQ.8 .AND. IABS(WK1-BK1).EQ.2) GOTO 345
        IF (BK2.GT.WK2) GOTO 310
        IF (BK2.EQ.WK2-2 .AND. WK1.EQ.BK1) GOTO 277
        IF (WK2.EQ.8 .AND. IABS(WK1-WP1) .EQ.2) GOTO 330
        IF ((WK2.EQ.7 .OR. WK2.EQ.8) .AND. BK2.EQ.WK2
       +  .AND. IABS(WK1-BK1).EQ.2)  GOTO 280
        IF (IABS(INTBK).LE.2 .AND. IABS(INTQSQ).LE.2) INCVAL=1
        IF (IABS(INTBK).LE.1 .AND. IABS(INTQSQ).LE.1) INCVAL=2
        IF (INTBK.EQ.0 .AND. INTQSQ.EQ.0) INCVAL=3
        GOTO 310
C
C       BK ON (VERTICAL) EDGE OF BOARD
C       ------------------------------
C
      288 IF (INTQSQ.EQ.-2) INCVAL=1
        IF (INTQSQ.GT.-2) INCVAL=2
        GOTO 290
C
C       WK ON SEVENTH RANK- TWO OR THREE FILES FROM THE PAWN
C       ---------------------------------------------------
C
      340 IF (INTBK.GE.-2) INCVAL=1
        IF (BK2.EQ.7 .AND. IABS(WK1-BK1).EQ.2) INCVAL=2
        IF (BK2.EQ.8 .AND. IABS(WK1-BK1).EQ.3) INCVAL=2
        IF (BK2.EQ.8 .AND. IABS(WK1-BK1).EQ.2) INCVAL=3
        IF (IABS(WK1-WP1).EQ.2 .AND. INTBK.GT.-2) INCVAL=INCVAL+1
        GOTO 290
```

```
C
C       WK ON SEVENTH RANK- TWO FILES FROM BK (ON THE EIGHTH RANK)
C       ---------------------------------------------------------
C
  345 INCVAL=2
      IF (IABS(WK1-WP1).EQ.3) INCVAL=3
      GOTO 290
C
C       BK TWO RANKS VERTICALLY BELOW WK
C       --------------------------------
C
  277 IF (INTQSQ.GE.-2) INCVAL=1
      GOTO 290
C
C       WK ON EIGHTH RANK- TWO FILES FROM THE PAWN
C       ------------------------------------------
C
  330 IF (INTBK.GE.-2) INCVAL=1
      IF (BK2.EQ.8 .AND. IABS(WK1-BK1).EQ.2) INCVAL=2
      GOTO 290
C
C       WK TWO FILES FROM BK ON THE SEVENTH OR EIGHTH RANK
C       -------------------------------------------------
C
  280 INCVAL=1
      GOTO 290
C
C       ADJUSTMENT FOR WK ON SIXTH RANK
C       -------------------------------
C
  310 IF (WK2.EQ.6 .AND. INTBK.GE.0 .AND. IABS (WK1-WP1).LE.2
     +   .AND. IABS(BK1-WP1).GT.1) INCVAL=INCVAL+1
C
C       **************************
C       TEST USING EFFECTIVE DISTANCE
C       **************************
C
  290 IF (MAXO(IABS(BK1-WP1),8-BK2)+INCVAL .GT. 9-WP3) GOTO 110
C
C       SPECIAL CASES- INCLUSIONS
C       -------------------------
C
  250 IF ((BK1.EQ.1 .OR. BK1.EQ.8) .AND. IABS(BK1-WP1).EQ.1
     +   .AND. IABS(BK1-WK1).EQ.2 .AND. WK2.GE.WP2  .AND. WP2.EQ.BK2)
     +  GOTO 110
       IF (WP2.EQ.2 .AND. BK2.EQ.2 .AND. SAMESD .AND. IABS(BK1-WP1).EQ.2
     +   .AND. ((IABS(WK1-WP1).EQ.1 .AND. WK2.EQ.6)
     +   .OR.    (WK1.EQ.BK1 .AND. WK2.EQ.5)
     +   .OR.    (IABS(WK1-WP1).EQ.3 .AND. WK2.EQ.4))) GOTO 110
       IF (WP2.EQ.2 .AND. BK2.EQ.3 .AND. WK2.EQ.5
     +   .AND. WK1.EQ.BK1 .AND. IABS(WK1-WP1).EQ.2) GOTO 110
C
C       PAWN CANNOT RUN
C       ---------------
C
  200 IRES=0
      RETURN
C
C       PAWN CAN RUN
C       ------------
C
  110 IRES=1
      RETURN
      END
```

References

Bramer, M.A. Forthcoming Ph.D. Thesis. 1977, The Open University.

Clarke,M.R.B. The Computational Complexity of King and Pawn Versus King. 1975, Mimeo. Forthcoming in Clarke,M.R.B.(ed.), Advances in Computer Chess 1. Edinburgh University Press.